MASTER EN SCIENCES INFORMATIQUES

Un environnement d'aide à la programmation d'un robot "La couche objet"

Berthet, Philippe

Award date:
1986

Awarding institution:
Universite de Namur

UN ENVIRONNEMENT D'AIDE A LA PROGRAMMATION D'UN ROBOT : "LA COUCHE OBJET"

Mémoire présenté par
Philippe BERTHET
en vue de l'obtention du titre
de licencié et maître en informatique.
Année académique 1985 - 1986.

Je tiens à remercier Monsieur Luc DEKEYSER, mon maître de stage, pour sa disponibilité à mon égard et pour ses judicieux conseils tout au long de l'élaboration de ce travail.

Mes remerciements vont aussi à Monsieur Axel VANLAMSWEERDE, promoteur de ce travail, dont les critiques consrtuctives ont considérablement contribué à la forme définitive de ce mémoire.

Je tiens également à remercier tous les membres du personnel du service d'informatique médicale à Louvain pour leur chaleureux accueil.

TABLE DES MATIERES.

0. INTRODUCTION.

Ce travail a été réalisé au département d'informatique médicale de l'hopital universitaire Gasthuisberg à Leuven. Initialement, il y avait lieu de réaliser un environnement d'aide à la programmation d'un micro-robot destiné à des travaux de laboratoires médicaux Cependant,au delà de cet objectif, il était possible d'élargir l'application à la manipulation d'objets quelconques.

Actuellement, la plupart des robots sont programmés au moyen de langages de très bas niveau, très proches de la mécanique et de l'éléctronique du robot. Par conséquent, les programmes sont long à élaborer et sont très peu portables puisqu'ils dépendent des caractéristiques des robots pour lesquels ils sont faits. Un langage de haut niveau devrait permettre à l'utilisateur de programmer une tâche complexe plus simplement en assurant une meilleure portabilité et une possibilité d'adaptation à des exigeances variées

Comme nous le verrons au premier chapitre, il existe plusieurs façons de programmer un robot mais, malgré les efforts qui ont été accomplis, il n'existe pas encore, dans ce domaine, de langage de programmation évolué.

Le projet dans lequel s'inscrit le présent mémoire cherche à mettre en oeuvre une programmation robotique de type évolué. Ce projet

s'est étendu sur 2 ans et à fait l'objet l'an dernier du mémoire de Claude Vanhemelryck et cette année de celui de Anne Dardenne. On trouvera une description succinte de ces recherches au chapitre 2.

Le présent travail tente de développer un niveau de programmation centré sur les objets que doit manipuler le robot. Il faut donc pouvoir représenter ces objets par un modèle que l'on pourra ensuite traiter informatiquement. Dans un premier temps (au chapitre 3), nous ferons un bref survol de ce qui existe en matière de modèles de représentation d'objets à 3 dimensions, ensuite (au chapitre4), nous essayerons de définir un modèle selon nos besoins et vis à vis de ce qui existe, et enfin (au chapitre 5), nous verrons comment manipuler, par des actions appropriées, les objets représentés dans le modèle.

1. INTRODUCTION A LA PROGRAMMATION ROBOTIQUE.

" A robot is a programable multifonction device designed to move material, part, tools or specialised devices through variable programmed motion for the performance of variety of task".

definition d'après le R.I.A (Robot Institute of America) in [NAG84].

Quoi qu'en dise cette définition, on appelle encore souvent "robot" un outil très spécialisé programmé une fois pour toutes pour une tâche spécifique . Cependant, la tendance actuelle semble vouloir rapprocher les robots de la définition pré-citée, ainsi désignant par ce mot un outil multifonction programmable pour une large variété de tâches.

Nous nous intéresserons plus spécialement dans ce travail aux robots connus sous le nom de bras articulés, c'est à dire bras munis d'au moins 3 articulations (épaule, coude et poignet) et terminés par une pince. Ce type de robot est le plus répandu et permet déja beaucoup d'applications différentes.

Il existe plusieurs manières de programmer un robot que l'on peut classer dans 2 grandes catégories : programmation ON-LINE et programmation OFF-LINE. Les méthodes les plus populaires et les plus

employées jusqu'à présent sont toutes on-line et n'utilisent donc pas de langage de programmation proprement dit.

1.1. PROGRAMMATION ON-LINE.

Il existe 2 types de programmation on-line :

1.1.1. Programmation par démonstration. (teaching by showing).

Une personne qualifiée pour éxecuter la tâche que le robot aura à faire saisit l'organe manipulateur du robot et éxécute avec lui les mouvements qu'il aura à reproduire. Pendant ce temps, le contrôleur enregistre plusieurs fois par seconde la position du robot. Il suffit alors de repasser séquentiellement par tous les points enrégistrés pour reproduire le mouvement désiré. Cette manière de programmer est probablement la plus répandue et la plus connue, (par exemple pour les opérations de peinture au pistolet) mais elle n'est sans doute pas vouée à un grand avenir à cause de ses fortes limitations et inconvénients tels que :

- Impossibilité de faire un listing du programme et de le corriger.

- Impossibilité d'employer des senseurs et donc d'enrégistrer des mouvements découlant d'opérations conditionnelles.

- Présence du robot obligatoire.

- Sophistication du matériel puisque le robot doit être capable de mémoriser les différentes positions de ses articulations lors de chaque échantillonage pendant l'apprentissage.

- Exigence pour le manipulateur humain de réaliser un mouvement parfait sous peine de tout devoir recommencer.

1.1.2. Programmation par boitier d'apprentissage.

Cette technique demande l'emploi d'un appareillage spécial qui permet au programmeur d'utiliser des boutons ou un "joy-stick" pour faire bouger le robot. Le programmeur, grâce à cette espèce de commande à distance, peut emmener l'organe manipulateur où il veut dans l'espace de travail du robot. Il le conduit donc aux différentes positions intermédiaires à atteindre lors de la réalisation du travail et enregistre chacune de celle-ci. Il suffit alors au robot de reproduire la séquence de positions intermédiaires enrégistrée pour effectuer le mouvement final demandé.

Bien que cette technique soit apparemment semblable à celle de la programmation par démonstration, elle comporte moins d'inconvénients au point de vue de la programmation proprement dite. En effet, beaucoup de constructeurs ont amélioré cette technique en y apportant des facilités bien utiles :

- Permettre au programmeur de spécifier l'attente d'un signal à chaque point où la position est enregistée. (de telle manière que l'on puisse synchroniser le robot avec un autre appareil de son environnement).

- Permettre de spécifier les points de passage où il doit s'arrêter.

- Spécifier la vitesse entre chaque point enregistré.

- Utiliser une librairie de sous-routines écrites par le vendeur et qui comporte des mouvements que l'on peut paramétriser et inclure dans un ensemble de mouvements que l'on a programmé. (par exemple, routine de paletisation, permettant de ranger ou enlever un ensemble de pièces sur une palette).

- Utiliser des senseurs simples (à valeurs binaires) grâce à l'emploi de routines pré-programmées qui permettent au robot d'éxecuter ou non un mouvement en fonction de la valeur donnée par le capteur.

- Disposer de simulateurs graphiques de robot permettant de programmer celui-ci avec feed-back visuel, à l'écran, de la simulation; ainsi le robot ne doit pas être mobilisé pendant la création du programme.

1.2. LA PROGRAMMATION OFF-LINE.

Depuis que les robots industriels existent, il y a toujours eu des langages pour les programmer, même si ces robots étaient prévus pour être programmés par démonstration ou par boîtier d'apprentissage comme c'est généralement le cas. Malheureusement, ces langages de programmation, conçus par les ingénieurs qui ont réalisé le hardware et toute la mécanique des robots, sont de très bas niveau. Ils offrent généralement un très petit ensemble d'instructions : ouverture et fermeture de la pince, test de senseurs, et une instruction speciale permettant de spécifier la position desirée pour le robot. Dans le cas d'un bras articulé, généralement composé d'une épaule, d'un coude et d'un poignet, l'instruction de positionnement du robot permettra de spécifier l'angle de rotation désiré pour chaque articulation afin de retrouver la position voulue.

Il va sans dire qu'un tel langage est extrêmement compliqué à utiliser tel quel; aucune visualisation n'étant possible. De plus, il est difficile pour l'utilisateur de penser en termes d'angle de rotation de chacune des articulations. Programmer une tâche complexe de cette façon est une entreprise hasardeuse. La complexité hardware et l'immense richesse mécanique et éléctronique offerte par les robots actuels (senseurs de toute sorte, stéréovision, complexité des articulations ...) face à la pauvreté des moyens sofware pour les exploiter font un peu penser à un ordinateur ultra sophistiqué muni d'un vaste ensemble de périphériques et ne disposant que d'un faible langage machine, sans même un système d'exploitation.

Heureusement, depuis quelques années, les chercheurs se sont rendu compte que la programmation on-line n'était pas suffisante pour permettre d'exploiter toutes les ressources de la robotique actuelle et se sont alors tournés vers des langages de programmation plus évolués qui sont maintenant en plein essor.

Le premier langage robotique un peu évolué à été développé par Unimation pour robot PUMA et est sorti sur le marché en 1982. Ce langage, VAL [NAG84] , offre un ensemble de primitives de programmation dans un langage de type Basic, et un boîtier d'apprentissage pour définir les points utiles dans l'espace de travail du robot. Depuis lors, plusieurs langages de même type ont été développés, mais ils ont tous pas mal d'inconvénients, notamment celui de ne pas pouvoir faire de calcul dans l'espace.

Une deuxième génération de langages est ensuite apparue avec notamment RAIL développé par Automatix et AML d'IBM [NAG84]. Ces langages se veulent proches des langages de programmation courants et offrent à l'utilisateur un moyen de faire de la programmation "structurée" (langages de type Pascal). Tous ces langages sont néanmoins encore assez difficiles à employer car l'utilisateur doit toujours penser en termes de monde tri-dimensionnel et programmer en fonction de la pince (pour prendre un objet, il ne faut pas penser à prendre l'objet mais à positionner la pince pour pouvoir prendre l'objet). C'est pour cette raison que ces langages sont d'un niveau appelé niveau "manipulateur". [LAT84].

Les problèmes rencontrés par la programmation en un langage de niveau "manipulateur" ont donné lieu au développement d'environnements de programmation - tels que par exemple des outils de mise au point interactive - ou à l'intégration de techniques de programmation par démonstration, ou encore à la connexion de bases de données. Malheureusement, la plupart de ces aides à la programmation n'apportent une aide effective qu'à partir du moment où le monde dans lequel travaille le robot à été représenté convenablement; ce qui, comme on le verra par la suite, pose un probleme particulièrement épineux.

Une approche plus générale pour résoudre les difficultés de programmation en robotique consiste à développer des langages de niveau "tâche", c'est à dire des langages tels que la description d'une tâche est orientée "objet" plutôt qu'orientée "robot" (penser à prendre l'objet plutôt que penser à orienter la pince pour prendre celui-ci). Un programme de niveau "tâche" sera donc constitué d'une séquence de buts décrits en termes

de relation spatiale entre objets que l'interpréteur traduira automatiquement en un programme de niveau "manipulateur". Par exemple, si l'on veut empiler 2 boites A et B, on le spécifiera au niveau "tâche" par une instruction du genre : "POSER A SUR B" ce qui se traduira au niveau "manipulateur" en une séquence d'instructions centrée sur la pince comme par exemple: DEPLACER PINCE EN [Coordonnées de la pince pour saisir A].

FERMER PINCE.

DEPLACER PINCE EN [Coordonées de la pince de telle sorte que A soit sur B].

OUVRIR PINCE.

Deux langages de niveau "tâche" ont été définis : AUTOPASS et LAMA [LIE77] (le manuel initial de AL contenait également un ensemble de primitives de niveau "tâche" qui n'ont jamais été implémentées). Lors du développement de ces deux langages, les chercheurs se sont rendu compte qu'il y avait de très grosses difficultés à traduire un programme de façon automatique d'un niveau à l'autre, surtout en ce qui concerne la recherche d'un parcours idéal entre deux points (path finding) et la saisie automatique des objets (grasping). Ces problèmes sont tellement aigus qu'ils ne sont toujours pas résolus actuellement et que les deux langages précités ne sont jamais sortis sur le marché avec les spécifications définies au départ. Ils ressemblent d'ailleurs plus dans leur forme actuelle à des langages de niveau "manipulateur"qu'à des langages de niveau "tâche".

2. LE PROJET.

Notre projet consistait à développer un environnement de programmation robotique de niveau élevé, ceci à partir d'un robot très commun : il s'agit d'un bras articulé ayant une pince comme organe manipulateur. Au départ, ce robot est programmable grâce à un langage de très bas niveau offert par le constructeur. Ce niveau de programmation qui est appelé le niveau "robot" est explicité au point 2.1.

En 1985, Claude Vanhemelryk s'est chargé de construire la première couche de programmation au dessus du niveau "robot" [VAN85]. Il s'agit de la couche "XYZ", dont le but principal est de permettre à l'utilisateur de faire abstraction presque complètement de l'aspect mécanique du robot en offrant un niveau de programmation comparable à celui que nous avons appelé précédemment niveau "manipulateur". Les concepts offerts par cette couche seront présentés au point 2.2.

Ce niveau de programmation s'est prolongé par l'adjonction d'une nouvelle couche : la couche "objet". Elle a été développée cette année et constitue le point central de ce travail.

Parallèlement à l'élaboration de la couche "objet" Anne Dardenne a construit une couche terminale qui fournit un niveau de programmation permettant de gérer la coordination de plusieurs robots travaillant simultanément dans un espace de travail commun, chacun d'eux étant dirigé par une programmation de niveau "objet" [DAR86]. Ce niveau ne

sera pas détaillé plus profondément puisqu'il se greffe immédiatement au dessus du niveau "objet"; nous n'en avons pas besoin pour développer l'application qui nous concerne.

2.1 LE NIVEAU ROBOT.

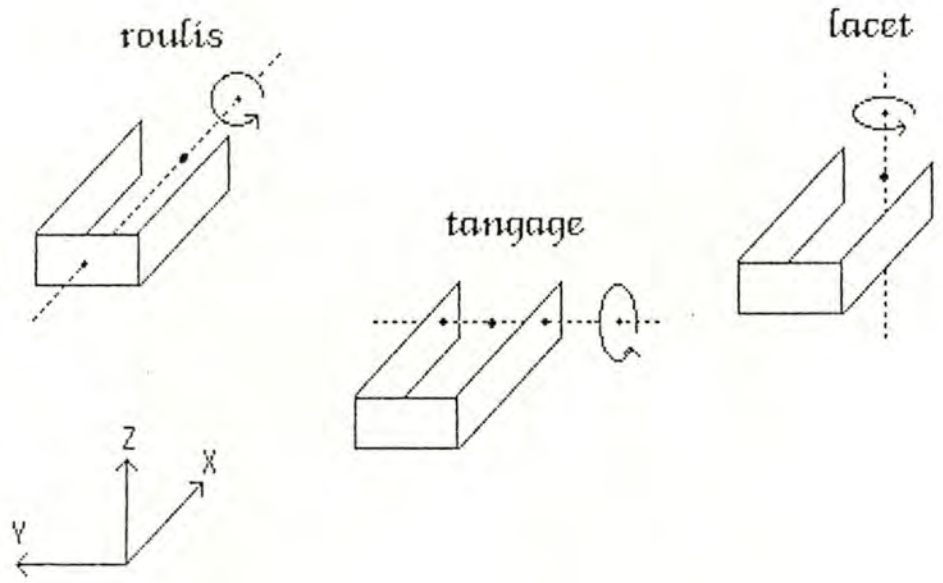
Le niveau robot, également dénommé niveau articulaire, est le niveau de programmation le plus élémentaire et correspond le plus souvent au langage offert par le constructeur. Il est constitué d'une série d'instructions de très bas niveau du même genre que celles que nous avons décrites précedemment; par exemple, une primitive importante est celle qui permet de placer le robot dans la position voulue en specifiant l'angle de rotation de chaque articulation composant le bras. Une spécification complète de ces primitives est disponible dans les manuels d'utilisation des bras articulés (et notamment dans le manuel du Mitsubishi Movemaster RM 501 qui a servi de modèle d'étude pour ce travail). On trouvera un bel exemple de programme résolvant le problème des tours de Hanoï qui utilise des primitives de ce niveau en annexe du mémoire de Claude Vanhemelryck [VAN85].

2.2. LE NIVEAU "XYZ".

Ce niveau fut développé et implémenté par Cl. Vanhemelryck. La finalité du travail consistait à réaliser un niveau de programmation que nous avons précédemment appellé niveau "manipulateur". Les primitives offertes à ce niveau sont articulées autour de trois concepts importants que nous rappellerons brièvement : le T.C.P., les tubes et les circuits.

2.2.1. LE T.C.P.

A ce niveau, l'utilisateur voit l'espace de travail du robot comme un espace tri-dimensionnel muni d'une base ortho-normée. Pour faire atteindre un point par l'organe manipulateur du robot, il suffit de lui donner la coordonnée cartésienne de ce point par rapport à un repère de référence plutôt que de lui donner l'amplitude des rotations de chacune de ses articulations. A cet effet, le concept de T.C.P (Tool Center Point) a été introduit. Le TCP désigne le point central entre les deux membres de la pince de l'organe manipulateur. On lui associe 6 paramètres; les 3 premiers désignent la coordonnée cartésienne du point cible, c'est-à-dire les 3 translations suivant les 3 axes du repère, tandis que les 3 derniers désignent les rotations que doit subir la pince pour atteindre l'orientation désirée. Ces 3 rotations s'appellent Tangage, Roulis et Lacet et sont explicitées dans la figure suivante :



Ces 6 paramètres correspondent aux 6 degrés de liberté "idéaux" que doit posséder un robot : 3 degrés de liberté translationnels (quand l'organe manipulateur peut se déplacer librement dans les 3 dimensions de l'espace), et 3 degrés de liberté rotationnels (quand l'organe terminal peut subir les 3 rotations suivant les 3 axes du repère de référence). Il est à noter que beaucoup de robots ne possèdent que 5 degrés de liberté, le lacet étant souvent absent et donc imposé par l'orientation du reste du bras lors des déplacements translationnels pour atteindre le point désiré.

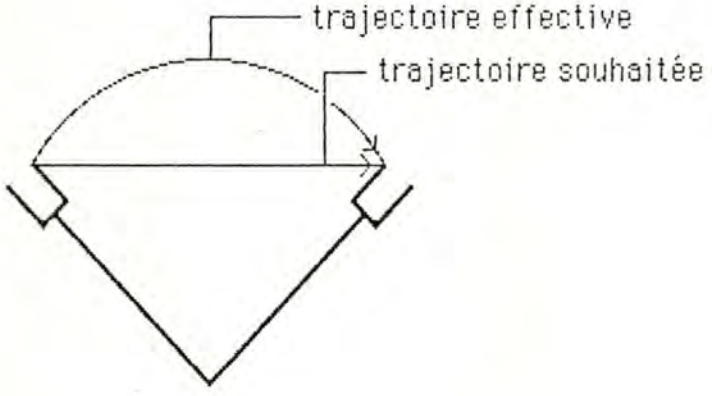
On peut déja sentir à ce niveau le progrès dans la facilité de programmation; en effet, l'utilisateur peut, dès à présent, faire abstraction presque totale du bras articulé et doit beaucoup moins se soucier de la manière dont celui-ci est construit (nombre d'articulations, articulations par simple charniere ou rotule, ...). Seul est à considérer l'organe terminal et la manière dont il est positionné et orienté dans l'espace de travail.

Les primitives de niveau "XYZ" agissant sur le TCP sont par exemple "TRANSFORMATION" qui calcule et renvoie le TCP de la pince à partir de la position articulaire du robot ou "TRANSFORMATION INVERSE" qui en est le pendant.

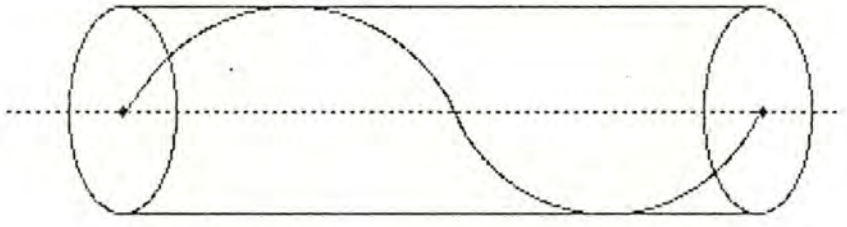
Outre le TCP, le niveau "XYZ" offre d'autres concepts qui peuvent être importants. Parmi ceux-çi, se trouve le concept de tube qui se révélera bien utile pour le niveau objet.

2.2.2. LES TUBES.

Pour un programmeur de niveau "XYZ", le déplacement le plus naturel de la pince entre 2 points sera évidemment la ligne droite puisque le monde dans lequel il travaille est un espace euclidien à trois dimensions. Toutefois, au niveau articulaire, la trajectoire la plus naturelle pour relier deux points sera une composition d'arcs de cercle puisque le mouvement sera effectué par une série de rotations suivant les articulations appropriées. Lors de la conversion automatique d'un programme de niveau "XYZ" en instructions de niveau articulaire, il se pourrait qu'un déplacement à effectuer par la pince soit traduit par une trajectoire tout à fait inattendue.



Il est donc indispensable de résoudre ce problème en offrant au programmeur de niveau"XYZ" un moyen de contrôler la trajectoire de la pince. Pour cela, on a créé la notion de tube. Un tube est défini par deux points (qui sont les points de départ et d'arrivée de la pince lors du mouvement), et par un certain diamètre (appellé tolérance). Lors d'un mouvement dans un tube, il est garanti que la pince ne sortira pas de l'enveloppe definie par le diamètre de celui-ci. Un mouvement dans un tube est donc une succession de mouvements circulaires à l'intérieur du tube. Il apparaît alors que plus le diamètre du tube sera petit, plus il y aura de mouvements circulaires pour le parcourir et moins le parcours sera performant et rapide, mais plus le mouvement sera rectiligne.



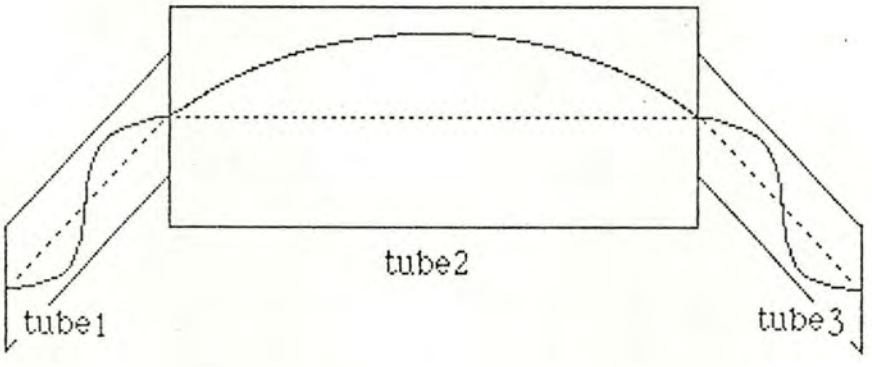
Le programmeur de niveau "XYZ" doit donc trouver un compromis entre la précision qu'il veut atteindre et la performance de son mouvement. Un mouvement dans un tube est donc defini par : deux TCP (positions de départ et d'arrivée de la pince lors du mouvement), un diamètre et la vitesse de déplacement maximum de la pince dans le tube.

A ce niveau, on trouve, par exemple, comme primitive concernant les tubes: "PARCOURIR-TUBE" qui provoque, à une vitesse maximum donnée, le déplacement du TCP de la pince sur le TCP cible selon une trajectoire inscrite dans l'enveloppe cylindrique que représente le tube.

Si ce tube a comme diamètre une valeur conventionnellement fixé à l'"infini", alors la trajectoire sefera selon un enchaînement minimum de rotation des articulations du bras, c'est-à-dire suivant le mouvement le plus naturel pour le niveau articulaire.

2.2.3. LE CIRCUIT.

Le niveau "XYZ" possède également le concept de "circuit" qui est défini par une trajectoire composée d'un ensemble de tubes. Ce concept peut être utile lorsque dans un parcours, il apparaît que certains passages sont plus critiques que d'autres et demandent une trajectoire plus rectiligne. On pourra dès lors définir pour ces passages un tube de diamètre plus étroit, ce qui provoquera un déplacement plus précis.



2.3. LE NIVEAU OBJET.

Le niveau de programmation objet est celui qui sera développé dans ce travail. Il correspond en gros à ce que Latombe [LAT84] appele le niveau "tâche".

Le niveau "XYZ", malgré un certain progrès obtenu en ce qui concerne les facilités de programmation par rapport au niveau robot, n'offre pas encore une programmation que l'on appele traditionnellement de type évolué. Pourtant, comme le dit Lambird [LAM83], il y a beaucoup de raisons d'employer un langage évolué dans la programmation de robots. Notamment celle de permettre à l'utilisateur de décrire le problème et sa solution d'une manière effective et naturelle et donc de donner l'occasion de se concentrer sur des concepts de haut niveau plutôt que sur des détails d'implémentation. En progammation "objet", le programmeur devra pouvoir spécifier ses actions en termes de buts à atteindre par rapport aux objets qui définiront son environnement et avec lesquels il devra travailler, plutôt que, comme l'exigeait la programmation en "XYZ", en termes de buts à atteindre par rapport à des positions spécifiques de la pince. Par exemple, si l'action à éxécuter consiste à prendre une boîte, le programmeur spécifiera seulement un ordre du genre "PRENDRE BOITE_A" sans être obligé de positionner la pince à une coordonnée prédéfinie et d'envoyer l'ordre de fermer la pince. La programmation "objet" parait évidemment beaucoup plus naturelle car lorsqu'un homme prend une boîte, il ne pense certainement pas à positionner sa main dans l'espace de telle manière à pouvoir prendre la boîte mais simplement au but terminal de son mouvement, c'est-à-dire saisir la boîte.

D'autre part, une telle programmation peut être qualifiée d'évoluée puisque les concepts que manie le programmeur sont de haut niveau et servent à résoudre un problème par une voie naturelle correspondant à des actions de base que développerait n'importe quel être humain pour résoudre le problème manuellement. La formulation d'une tâche à ce niveau consiste dès lorsà préciser uniquement les mouvements que les objets doivent subir : le programmeur décrit donc la tâche à accomplir et non plus la manière dont il faut l'accomplir.

2.3.1. MODELISATION DE L'ESPACE DE TRAVAIL DU ROBOT.

Pour arriver au niveau de programmation "objet", il faut évidemment enrichir l'environnement de programmation "XYZ".

Le niveau "XYZ" se contentait de représenter l'espace de travail du robot par un espace euclidien à 3 dimensions muni d'une base ortho-normée. Nous avons maintenant à tenir compte des objets qui se situent dans cet espace et à les y représenter pour que l'utilisateur puisse les employer dans sa programmation. Il faut donc pouvoir introduire une représentation de l'ensemble des objets ainsi que la situation de chacun d'eux dans l'espace de travail. Cette "connaissance" sera stockée dans une base de données qui sera mise à jour à chaque déplacement d'objet ou à chaque modification dans la forme d'un objet (lors d'un assemblage par exemple).

A ce niveau deux problèmes se posent. D'abord, il s'agit de choisir un modèle de description d'objets à 3 dimensions qui soit adéquat et performant, c'est à dire qui permette de représenter les objets et le monde

de travail du robot le plus précisément possible tout en restant simple; ensuite il faut trouver un moyen de gérer cette description en mémoire suivant le modèle choisi. Ces 2 problèmes seront respectivement traités aux sections 4.1 et 4.2.

2.3.2. LE LANGAGE DE PROGRAMMATION.

La difficulté réside ici à trouver un langage simple, naturel et complet. Comme nous l'avons assez répété, il faut que le langage puisse exprimer la tâche à accomplir de la manière la plus proche possible de celle d'un opérateur humain qui devrait décrire le travail qu'il a à faire. Pourtant, il existera toujours un dilemme entre la simplicité et la complétude du langage offert. En effet, le langage doit posséder un vocabulaire assez complet pour permettre de décrire tous les types de tâche, mais en même temps il ne peut pas être trop lourd à utiliser ni trop compliqué à implémenter (par exemple, des actions telles que visser, couper, enfoncer, ... peuvent parfois être utiles à certaines tâches spécifiques mais trop particulières pour être implémentées).

La politique choisie dans ce travail a été d'essayer d'implémenter une espèce de "boîte à outils" où les outils de base seraient très simples et peu nombreux tout en permettant d'exprimer un maximum de choses. Comme le niveau objet doit permettre de spécifier des actions sur les objets, les actions seront calquées sur les différents concepts du modèle de description des objets à 3 dimensions.

Un autre problème que pose toute programmation robotique est la capacité de gérer des informations incertaines ou erronnées..

L'incertitude et l'erreur dans la position des objets et du robot sont inévitables. Chaque articulation du robot n'a qu'un certain degré de précision, de même que l'emplacement d'un objet dans l'espace. Ces erreurs que Brady [BRA85] appelle "erreurs de base" se cumulent. Toute approche de la manipulation robotique doit donc réduire l'incertitude à un niveau acceptable ou être assez robuste pour travailler correctement en dépit de l'incertitude. Lozano-perez [LOZ85] prétend que le seul moyen de traiter l'incertitude lors d'assemblages de haute precision est d'employer des capteurs sensitifs et d'utiliser les forces générées par le contact du manipulateur avec des objets pour obtenir des informations à propos de la position relative du manipulateur par rapport à ces objets. La trajectoire du manipulateur devient alors une fonction de ces différentes forces. toutefois, le fait que les modèles de représentation ne peuvent pas être exactement fidèles à la réalité n'est pas une raison suffisante pour les rejeter. Cela signifie seulement que les utilisateurs doivent prendre en compte les erreurs inévitables et que l'on ne peut pas demander des tâches qui exigent plus de précision que celle du système de contrôle du manipulateur.

3. MODELISATION DES OBJETS : REPRESENTATION DE SOLIDES EN TROIS DIMENSIONS.

Nous allons brièvement explorer ce qui existe aujourd'hui en matière de représentation d'objets en 3 dimensions et essayer de faire un choix parmi les différents modèles.

3.1 LES MODELES DE REPRESENTATION.

3.1.1. Représentation par le contour des formes (Wire-frame representation).

[REQ80],[BES85],[PER83] [BRA84].

Ce modèle de représentation est encore couramment employé en CAD (ComputerAid Design) et est basé sur les modèles de représentation d'objets à 2 dimensions. C'est la projection de chacun des objets sur un plan qui est représentée, ceci par un ensemble de couples de points qui constituent chacun une arête de l'objet à représenter. Cette méthode est fortement critiquée car elle est ambigüe. En effet, une même représentation peut conduire a considérer plusieurs interprétations de la structure d'un objet ou de son orientation dans l'espace.

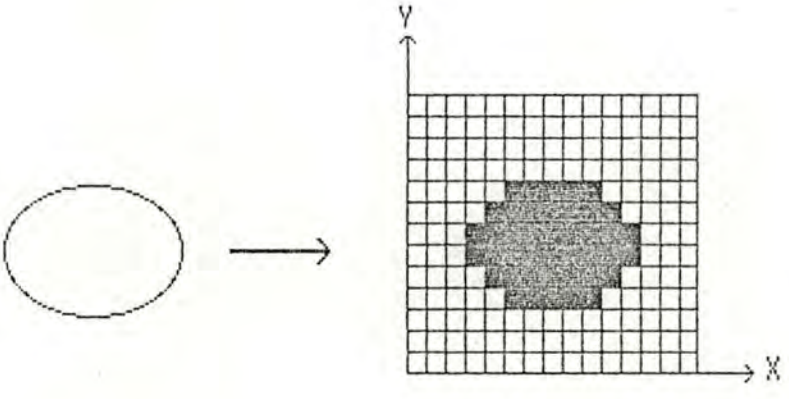
3.1.2. Représentation par occupation spatiale.

Cette méthode représente les objets en définissant des sous-régions de l'espace à 3 dimensions occupé par ces objets. Il existe plusieurs représentations de ce type qui sont communément utilisées :

a. Représentation par volumes élémentaires (Voxel rep).
[REQ80],[BES85].

Les objets sont representés par une liste de volumes élémentaires de l'espace occupé par l'objet. Ces voxels (volume elements) sont généralement des cubes de taille fixe représentés par une coordonnée spatiale déterminant le centre du cube. La seule relation qui existe entre deux éléments consécutifs de la liste est une relation de contigüité spatiale de deux cubes élémentaires de l'objet représenté. Les algorithmes manipulant une telle structure de données sont assez simples mais cette représentation utilise beaucoup d'espace memoire. (surtout lorsque l'on veut avoir une description précise puisque il faut alors mettre en oeuvre des voxels de petite taille.)

Exemple : (en deux dimensions)



Pour affiner la représentation, on a parfois utilisé des "voxels" de forme plus complexe; par exemple Faugeras [FAU84] propose d'utiliser comme volumes élémentaires des polyèdres dont la forme en 3 dimensions est équivalente à un hexagone en 2

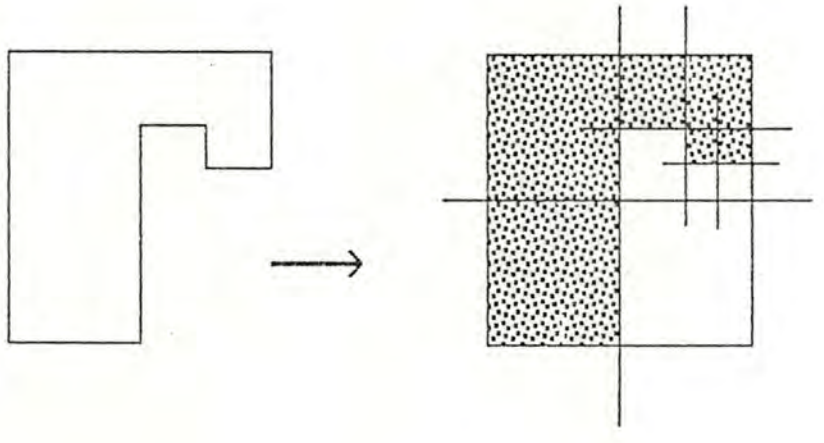
dimensions et qu'il a baptisé du nom tout à fait charmant de "Rhombododécahédron".

b. Représentation par arbre à 8 branches (Octree rep).

[BES85],[YAU83]

Un octree est une représentation arborescente de l'occupation spatiale d'un objet. La racine de l'arbre représente le plus petit cube contenant entièrement l'objet a modéliser. Ce cube est alors divisé en 8 autres cubes d'égale dimension qui subissent eux aussi le même traitement, et ainsi de suite jusqu'à ce que chaque cube fils obtenu par décomposition du cube père soit entièrement plein ou entièrement vide. Ces cubes fils forment alors les feuilles de l'arbre de représentation. Un volume est donc représenté par des cubes de taille différente, taille qui dépend de la distance du noeud qui le représente à la racine de l'arbre. Cette représentation est plus avantageuse en taille mémoire et en temps cpu de par la plus grande richesse de la structure de données mais les algorithmes sont un peu plus compliqués.

Exemple : (en 2 dimensions)



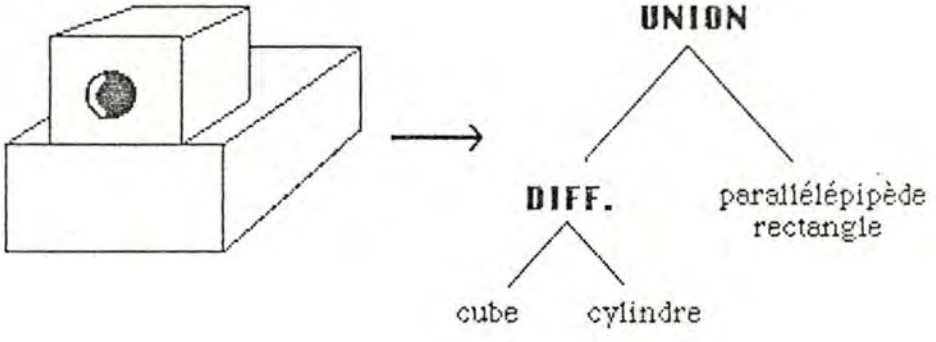
c. Représentation par décomposition en volumes tétraédriques.
(Tetrahedral cell decomposition representation). [BES85].

Cette méthode propose une décomposition d'un espace a 3 dimensions en tétraèdres de manière analogue à la décomposition d'une surface plane (2 dimensions) en triangles. Elle est surtout employée pour des applications mathématiques.

3.1.3. Géometrie constructive de solides. (Constructive Solid Geometry : CSG). [REQ80],[TIL84],[BES85]

Un objet en CSG est représenté en termes d'un ensemble de volumes primitifs (cubes,cylindres,cônes,spheres,...) et d'un ensemble d'opérateurs ensemblistes (union, intersection, différence). La structure de données correspondante est un arbre binaire où les noeuds terminaux représentent des instances de volumes primitifs et les noeuds intermédiaires représentent des opérateurs et des informations de position spatiale. Les arbres CSG permettent de définir des objets de manière non ambigüe avec très peu d'informations. Cependant, les algorithmes qui les manipulent sont assez lourds et les objets dont les surfaces sont accidentées sont presque impossibles à représenter.

Exemple :



3.1.4. Représentation par balayage. (Sweep rep. or generalized cone.)
[BES85],[PER84].

a. Par balayage translationnel.

On definit les objets par une surface à 2 dimensions et par un vecteur perpendiculaire à la surface considérée. Le volume de l'objet est défini par l'espace balayé par la surface lors de sa translation selon le vecteur qui lui est perpendiculaire.

b. Par balayage rotationnel.

L'idée est similaire mais c'est la rotation de la surface initiale autour d'un axe prédéfini situé dans le même plan qui définit le volume occupé par l'objet.

Ces modèles ne sont pas souvent utilisés car ils ne peuvent représenter qu'un nombre restreint d'objets (objets symétriques et uniformes). Toutefois, d'après Tilove, à partir de l'idée de mouvement initial d'une surface , on peut représenter plus facilement le mouvement de l'objet dans l'espace et ce concept est donc assez commode lors de la recherche de trajectoires avec détection de collisions.

3.1.5. Par représentation des surfaces. (Surface boundary rep or B-rep)
[REQ83],[BES85],[BOY79],[LOZ79].

Un objet est représenté par un polyèdre qui l'enveloppe. Chaque face du polyèdre est elle-même représentée soit par l'équation du

plan dans lequel elle se trouve, soit par décomposition de chaque face en segments de droite qui sont eux-mêmes représentés par les coordonnées des 2 points qui marquent leurs extrémités. Des techniques plus avancées permettent de modéliser des surfaces courbes (par des équations mathématiques) qui délimitent le volume de l'objet à représenter. La B-rep pose certains problèmes lors du traitement d'objets creux ou troués mais est efficace et souple pour les objets pleins (par exemple si l'on veut savoir si un point quelconque de l'espace se situe à l'intérieur d'un objet ou si l'on veut déterminer le volume occupé par un objet).

3.1.6. Représentations hybrides. [REQ80]

a. CSG / Surface.

Les représentations sont faites avec des arbres de genre CSG dont les feuilles terminales sont aussi bien des solides primitifs que des volumes non primitifs représentés par B-rep.

b. CSG / balayage.

Les représentations sont également faites avec des arbres de genre CSG dont les feuilles peuvent être des solides non primitifs représentés par sweep-rep.

3.2. GEOMETRIC MODELING SYSTEM.

L'importance des recherches sur les modèles de représentation d'objets en 3 dimensions à provoqué le développement d'une branche qui l'englobe entièrement : le CAD/CAM (Computer Aid Design &

Manufacturing : robotique, computer vision, graphisme d'espace a 3 dimensions, ...).

La plupart des modèles actuellement employés dans les applications industrielles sont basés sur une représentation CSG ou B-rep (plus de 80% du marché) [REQ83]. il peut paraître curieux que les modèles d'occupation spatiale tels que les voxels-rep ou les octree-rep ne soient pas plus employés puisqu'ils sont sous une forme qui, à priori, paraît plus simple à manipuler par ordinateur. Ce phénomène tient au fait que le modèle de représentation interne n'est pas suffisant à lui seul pour pouvoir travailler sur des objets à 3 dimensions. Il faut que au modèle soit associé un langage dont la compilation produit la représentation interne de l'objet; il permettra à l'utilisateur de faire une description des objets à modéliser et de manipuler ces objets au sein du modèle. Un tel système (input language + modèle interne) s'appelle "Geometric Modelling System" (GMS).

Ces langages de description de solides à 3 dimensions sont encore pour la plupart de type procédural. L'utilisateur doit donner une description formelle des objets qu'il veut modéliser grâce à un langage spécifique et approprié, par exemple GDP (Geometric design processor) dans PALD-1 [LIE77] ou le langage défini par Grossman et utilisé dans AUTOPASS [GRO76]. Malheureusement, comme on va le voir, les moyens d'entrer les descriptions sont très faibles. Ces langages de description d'objets sont très complexes, difficiles à utiliser et donnent des résultats de représentation qui, par rapport à leur complexité d'emploi, sont loin d'être satisfaisants.

On peut s'étonner du peu de recherches effectuées à propos des techniques de description des représentations d'objets par rapport à

celles faites pour les modèles internes et du pessimisme presque unanime quant aux progrès dans cette direction. Plusieurs problèmes se posent en effet. D'abord, il est réellement très compliqué de donner une description formelle d'un objet en 3 dimensions et, plus la définition que l'on donne de l'objet est succincte, moins la représentation de l'objet est précise. Si l'on veut employer les modèles de représentation au mieux de leurs possibilités, il faut donner une description tellement complexe de l'objet qu'elle découragerait n'importe qui. D'autre part, l'association entre les objets à décrire et la description de leur situation et orientation dans l'espace est loin d'être évidente a réaliser. Tous les systèmes rencontrés jusqu'à présent sont lourds, compliqués à utiliser et extrêmement peu parlants pour l'utilisateur.

Le problème de l'introduction de l'information par rapport au traitement proprement dit que l'on veut y apporter n'est pas spécifique à la représentation des solides, c'est un problème inhérent a l'informatique d'aujourd'hui. Le directeur de la section informatique du CNRS francais déclarait en substance lors d'une émission de vulgarisation informatique : "L'ordinateur est un outil extraordinairement puissant quant à la capacité qu'il a de traiter et de sortir d'énormes quantités d'information à une vitesse déconcertante. Pourtant, il subsiste un inconvenient majeur : les systèmes d'entrée sont proportionnellement d'une lenteur catastrophique. Ce déséquilibre entre le flux d'entrée et de sortie est réellement un des problèmes majeurs de l'informatique actuelle."

Il apparaît donc clairement que la précision de la description des objets dans le modèle choisi est proportionnelle à la complexité de l'interface utilisateur. Il y a donc lieu de trouver un compromis entre ces extrêmes.

On comprend maintenant déja mieux pourquoi les modèles de type CGS ou B-rep occupent plus de 80 % du marché. Il est évidemment beaucoup plus facile pour un utilisateur de pouvoir décrire des objets par des instructions d'un langage orienté CSG que d'en donner une description permettant de créer une octree-rep ou une voxel-rep (le lecteur non convaincu essaiera de donner une desciption formelle d'un objet simple en découpant le cube qui l'enveloppe en sous-cubes jusqu'à ce tous soient pleins ou vides !).

Cependant, les modèles basés sur une octree-rep se developpent de plus en plus, ceci pour plusieurs raisons :

D'abord, l'octree-rep s'adapte particulierèment bien a certaines configurations hardware de par sa structure régulière (cablage d'instructions gérants des arbres à 8 branches), et des "octree machines" sont en passe d'etre commercialisées par Phoenix Data System.

Ensuite, de nombreuses recherches ont été faites sur la transformation automatique de représentation d'objets d'un modèle en un autre et nombre d'applications présentées actuellement sur le marché possèdent un GMS dont le modèle interne est different du modèle utilisé pour le langage de description d'objets. C'est le cas par exemple de PALD-2 dont le modèle interne est de type octree-rep et le langage de description de type CSG ou encore EUCLID (B-rep / CSG) ou ROMULUS (B-rep / sweep) [REQ80].

Enfin, les langages de description d'objets s'entourent de plus en plus d'aides et de pré-processeurs de toutes sortes qui allègent au maximum la partie procédurale de façon à rendre la description moins

rébarbative, plus précise et plus rapide. Outre les aides à interaction graphique qui sont de plus en plus répandues, il existe également des systèmes utilisant un laser pour mesurer les tailles des différents objets. Le laser, connecté à un système de visualisation graphique, permet à l'utilisateur de superviser le processus et le résultat de la construction du modèle [HAS79]. Un système interactif comparable, mais dont l'organe manipulateur lui-même sert d'outil de mesure, à été proposé par Grossman et Taylor en 79 (avec comme inconvénient l'immobilisation du robot). Enfin, le *nec plus ultra* en matière de description d'objets à 3 dimensions est sans aucun doute la stéréovision. Ce système permet d'obtenir automatiquement une description d'objet grâce à 2 caméras fixant un même point à partir de 2 endroits différents pour avoir une vue en 3 dimensions de l'espace observé [ALB84]. Malheureusement, bien que la stéréovision soit un domaine de recherche de pointe, les résultats sont difficiles à atteindre car les chercheurs butent sur des problèmes particulièrement ardus. En effet, la stéréovision demande une parfaite correspondance des deux caméras pour restituer une image à 3 dimensions à partir de 2 points de vue bidimentionnels différents. Les calculs que demande un tel travail sont énormes et prennent évidemment beaucoup de temps. (Il est reconnu que l'analyse et la digitalisation d'une seule image prise par une simple caméra demande à peu près une seconde à un processeur spécialisé).

4. LE MODELE UTILISE.

4.1. DESCRIPTION DU MODELE.

Aucun des modèles contenus dans la littérature ne convenait exactement à l'usage que l'on voulait en faire. Soit ils étaient trop compliqués à implémenter (comme par exemple les modèles utilisant le balayage ou la B-rep qui étaient trop mathématiques), soit ils étaient trop compliqués dans leur utilisation (CSG) ou impossibles à utiliser sans un bon langage de description d'objets qui aurait été long et difficile à réaliser (Voxel, octree).

Nous désirions un modèle utilisable immédiatement; donc qui possède un langage de description d'objets qui soit le plus simple possible pour pouvoir l'implémenter rapidement afin d'avoir accès au modèle le plus vite possible. En fait, nous n'avons pas dû implémenter de langage de description d'objets. Comme nous avons travaillé en Prolog, nous avons orienté la définition du modèle de telle sorte que les objets à représenter se traduisent en concepts proches de ceux employés dans Prolog (principalement des listes d'éléments); ceci afin que l'on puisse entrer directement les descriptions sous forme de prédicat Prolog. Et c'est donc Prolog qui nous a servi de langage de description d'objets.

De plus, nous souhaitions que les concepts employés dans le modèle soient proches de ceux offerts par la couche "XYZ" (TCP, Tube, Circuits) afin de pouvoir les exploiter au maximum et pour que le passage d'une couche à l'autre soit harmonieux.

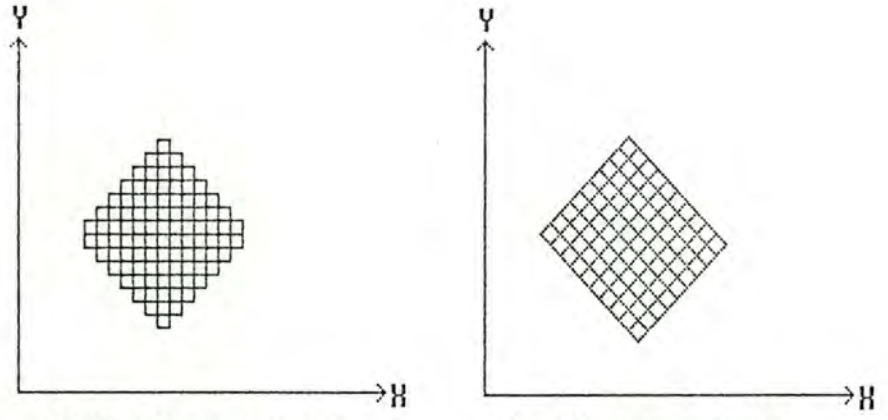
Enfin, nous souhaitions un modèle qui se prête à une décomposition hiérarchique dont les concepts du bas seraient simples et se complexifieraient de plus en plus en montant dans la hierarchie. Nous avons donc conçu un modèle hiérarchique tel que chaque niveau puisse fournir une description plus ou moins complexe et plus ou moins proche de la réalité des objets à représenter en fonction de la hauteur du niveau dans la hiérarchie. Chaque échelon de la hiérarchie comportera des éléments complexificateurs par rapport à l'échelon inférieur afin de donner une représentation un peu plus fidèle de la réalité. A chacun des niveaux de la hiérarchie viendront se greffer une série de primitives qui manipuleront les représentations du niveau considéré en utilisant les primitives des niveaux inférieurs. Ainsi, la construction du modèle et de son implémentation pourra se faire niveau par niveau en commençant par le bas, chacun des niveaux constituant, avec ses prédecesseurs, un "sous système utile".

Le modèle qui fut finalement choisi résulte d'un compromis entre la Voxel-rep et la CSG-rep. Voxel-rep parce qu'au bas de la hiérarchie on trouve des volumes élémentaires, CSG-rep parce que plus haut dans la hiérarchie, les volumes élémentaires se combinent pour former des volumes primitifs qui serviront à fabriquer des objets plus complexes suivant une méthode suggérée par le modèle CSG. Comme nous le verrons, ce compromis satisfait assez bien aux exigences que l'on vient de définir.

4.1.1 LES VOXELORS.

Chaque objet est représenté par un ensemble de cubes dont l'arête a une longueur unitaire. Un cube représente en fait le plus petit élément adressable par le robot dans le monde "XYZ". La différence entre ces

cubes et les voxels suggérés par la Voxel-rep est qu'ils sont orientés. Pour cette raison, nous avons appelé ces cubes particuliers VOXELORS (pour voxels orientés). Chaque voxelor est donc identifié par sa position et son orientation. Dans l'exemple (à 2 dimensions) ci-dessous, en voxel-rep, les carrés devront avoir leurs côtés parallèles aux axes ce qui n'est pas le cas en voxelors-rep puisqu'ils peuvent avoir n'importe quelle orientation.



Modélisation suivant la la Voxel-rep.

Modélisation suivant la Voxelor-rep.

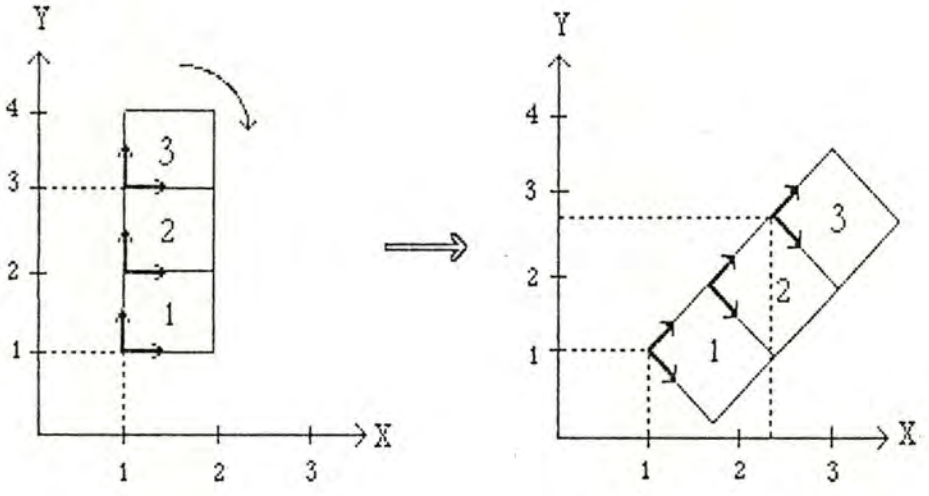
Avantages:

- Comme on peut le remarquer dans le dessin précédent, on aura souvent une définition plus précise qu'avec une modélisation par voxel-rep.

- Cette représentation s'ajuste très bien à la notion de TCP développé par Cl. Vanhemerlijk et qui est le concept principal retrouvé à la couche inférieure.

- A toute situation de la pince correspond un voxelor. En effet, pour le robot, l'espace de travail est un monde discret; il est

autour du voxelor n°1, la position de ce dernier sera toujours la même et son orientation sera définie par son ancienne orientation plus n unités suivant l'axe autour duquel il aura subit la rotation. Il n'en va pas de même pour les 2 autres voxelors qui auront subi une certaine translation qui leur aura fait atteindre une position non représentable. Par exemple le voxelor n°3 sera au point (2,3 ; 2,7) auquel ne correspond aucun TCP (et donc aucun voxelor) et nous serons forçés d'approximer sa position au TCP le plus proche (ici (2,3)).



4.1.2. LES OBJETS PRIMITIFS.

Pour pallier ces désavantages on va considérer un ensemble d'objets symétriques prédéfinis que nous appellerons objets primitifs (parallélépipèdes rectangles, cylindres, sphères, ...). Ces objets primitifs

seront en quelque sorte la représentation compilée d'un ensemble de voxelors de même orientation.

4.1.3. LES OBJETS.

Chaque objet du monde à représenter sera dès lors défini comme une composition de ces objets primitifs selon le même principe que celui employé dans la CSG-rep. On pourrait évidemment y retrouver toutes les opérations ensemblistes proposées par la CSG-rep (union, intresection, différence,...) suivant le degré de précision que l'on veut atteindre. Nous n'avons gardé dans le cadre de ce travail que l'opération d'union qui est sans aucun doute la principale et celle qui permet de représenter le plus facilement les objets.

4.1.4. LES OBJETS COMPLEXES.

On pourra ensuite, pour chaque objet, greffer des contraintes particulières à un objet (interdiction de certaines translations ou rotations par exemple pour un verre rempli d'eau que l'on ne peut pas pencher), ou des contraintes de dépendance entre deux ou plusieurs objets. Examinons les différentes contraintes que l'on pourrait porter sur un ou plusieurs objets :

- Contraintes sur un objet seul :

--> Objet inamovible

--> Objet libre ..... Contrainte de mobilité translationnelle

| Sur un axe
ou
| Sur un plan

..... Contrainte de mobilité rotationnelle

| Sur un axe

- Contraintes sur deux ou plusieurs objets provoquées par une relation entre eux :

--> Dépendance totale (Les objets sont fixés les uns aux autres : Assemblage).

--> Dépendance partielle (Les objets sont déposés les uns sur les autres : Montage).

4.1.5. LES SITES.

Il peut être utile de pouvoir permettre au programmeur de définir certains points particuliers de l'espace de travail du robot (points de saisie de certains objets, points de passage obligatoire lors de certains déplacements de l'organe manipulateur,...). Nous désignerons ces endroits spéciaux par le terme "site".

4.1.6. LES TUBES ET LES CIRCUITS.

De même, il est utile d'avoir les concepts de tubes et de circuits définis à la couche "XYZ" pour pouvoir effectuer des déplacements de la pince.

4.1.7. RELATIONS ENTRE LES DIFFERENTES COMPOSANTES DU MODELE.

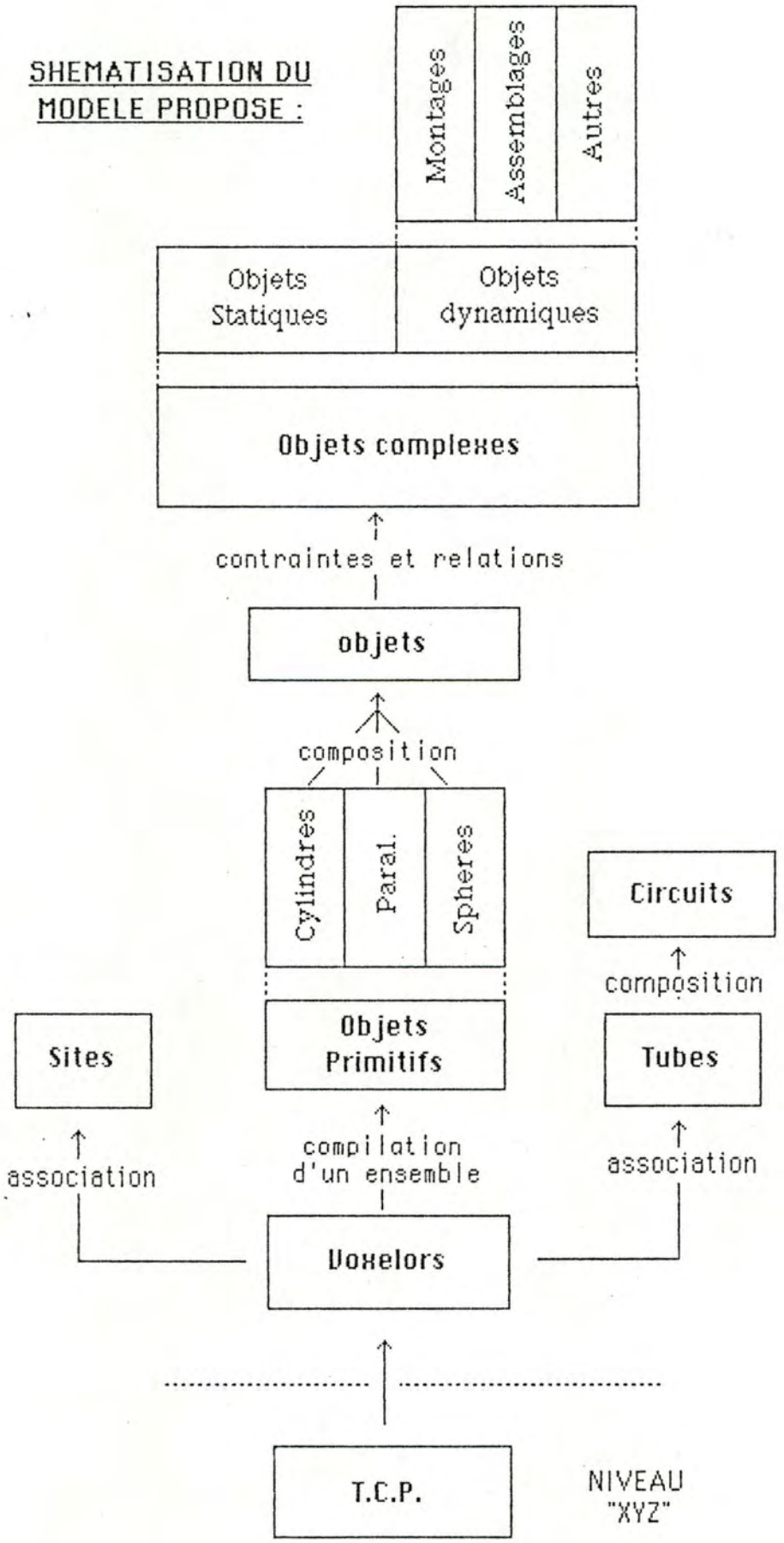
Le modèle que nous venons de définir correspond assez bien aux exigences que nous nous sommes fixés en début de chapitre. En effet, d'une part, il comporte bien des concepts proches de ceux offerts par la couche "XYZ" (voxelor, tube, circuit) , ensuite il adopte effectivement pour représenter les objets, une structure hiérarchique telle que nous la souhaitions (avec une relation de complexité croissante entre les niveaux). Le niveau le plus bas englobe le concept de voxelor, puis, le niveau suivant contient les objets primitifs de différentes formes qui sont des ensembles de voxelors de même orientation, ensuite, la composition de ces objets primitifs nous fournit les objets qui définiront le niveau suivant, et enfin, nous pouvons y greffer des contraintes pour parvenir au dernier échelon de la hiérarchie contenant les objets complexes (qui seront soit des objets statiques, soit des objets dynamiques, ces derniers pouvant être par exemple des assemblages ou des montages). Notons qu'un même objet peut être décrit à différents niveaux de la hiérarchie, et que la précision de sa description sera fonction du pallier dans lequel il est défini.

Les tubes, les sites et les circuits sont des concepts quelque peu marginaux dans ce modèle. Ils n'ont pas à s'inscrire dans la décomposition hiérarchique vue précédemment car ils sont indépendants de

la description des objets de l'espace de travail. Ces concepts utiliseront néanmoins ceux de plus bas niveau de la hiérarchie puisque, comme on le verra dans la représentation du modele, un site sera associé a un voxelor et un tube sera notamment composé de deux voxelors. Enfin, la relation qui existe entre les circuits et les tubes est une relation de composition, un circuit étant defini par une suite de tubes.

On trouvera a la page suivante une shématisation du modèle muni des relations qui viennent d'être expliquées.

SHEMATISATION DU
MODELE PROPOSE :

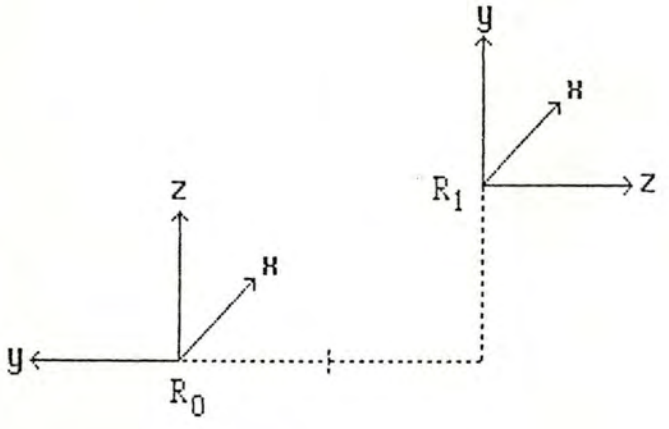


Exemple de représentation d'un repère R_1 par rapport à une base R_0 :

La situation de R_1 est donnée par :

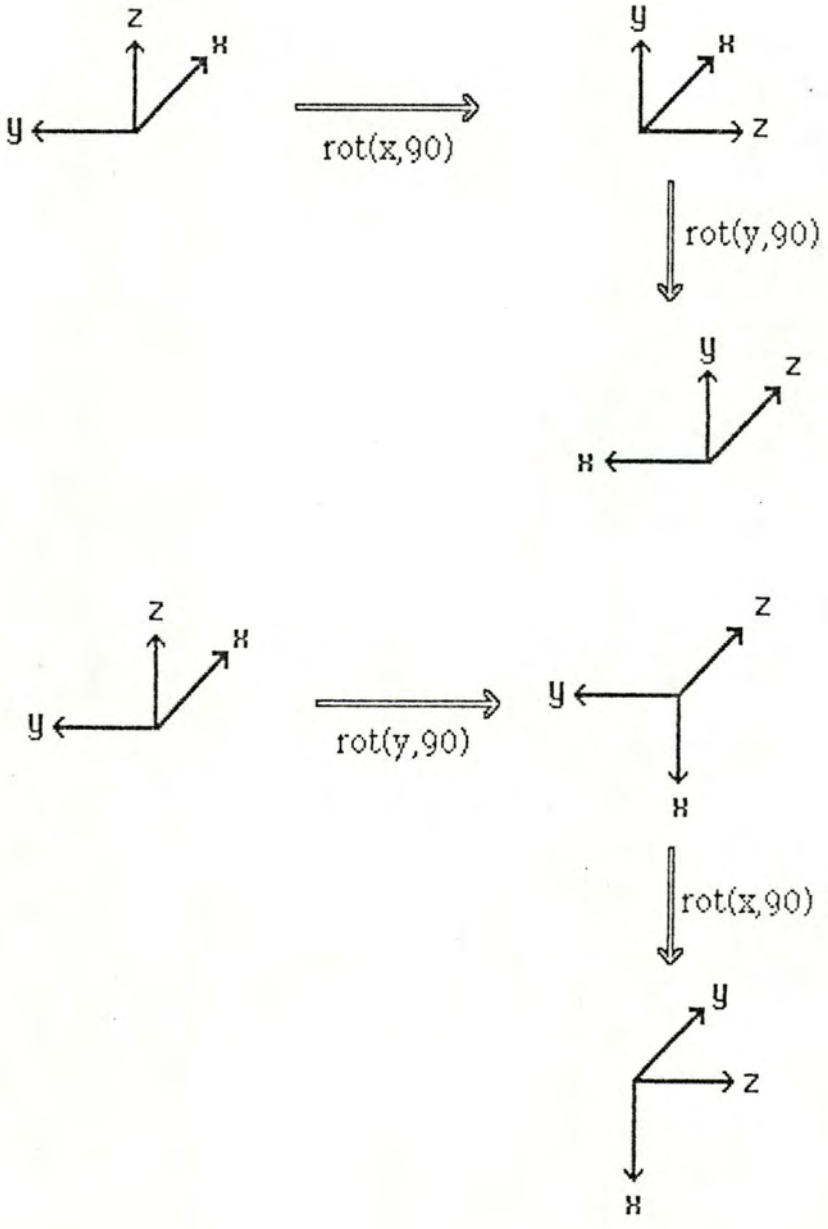
sa position : translation suivant X : 0.
Y : -2.
Z : +1.

et son orientation : roulis : +90°.
tangage : 0°.
Lacet : 0°.



Remarques :

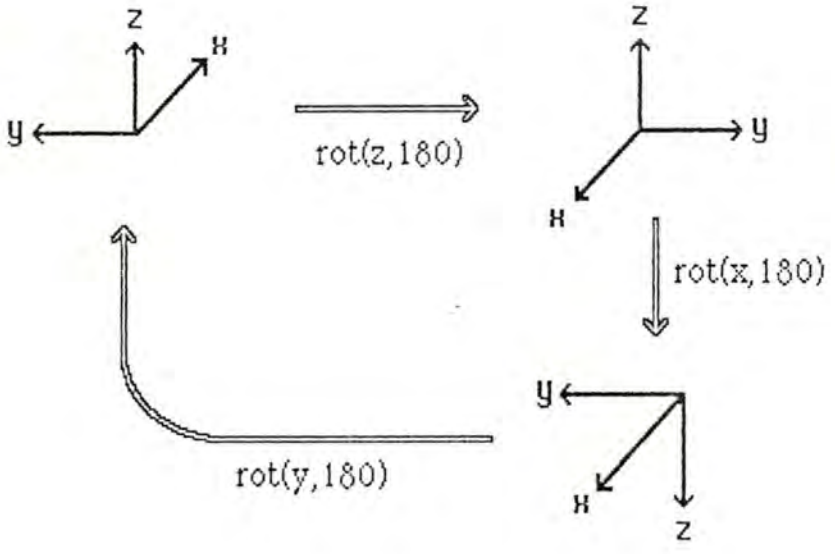
1. L'ordre dans la séquence de rotations est déterminant de la position finale. En effet, la position obtenue à la suite d'un roulis de 90 degrés (rot(x,90)) suivi d'un tangage de 90 degrés (rot(y,90)) sera différente de celle obtenue apres la séquence inverse (rot(y,90) puis rot(x,90)):



Il est donc primordial de fixer dès à présent une convention pour la séquence de rotations. Nous prendrons l'ordre suivant :

1. Rotation autour de l'axe des Z (tangage).
2. Rotation autour de l'axe des X (roulis).
3. Rotation autour de l'axe des Y (lacet).

2. La situation d'un repère n'est pas identifiant d'un sextuplet de paramètres. En effet, deux séquences de rotations différentes peuvent conduire à la même situation puisque, d'après les observations, on remarque que : (T,R,L) = (T+180, R+180, L+180).

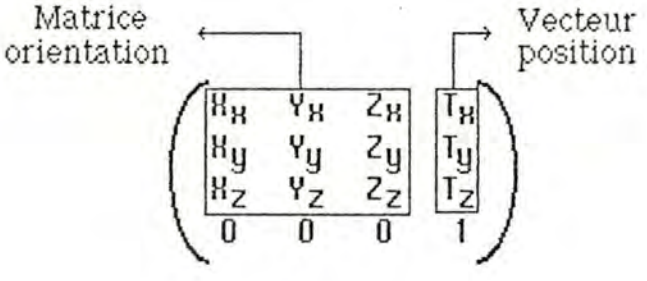


3. Une telle représentation d'une base ortho-normée est en harmonie avec les concepts de la couche "XYZ". En effet, les trois translations et les trois rotations correspondent au déplacement et au tangage, au roulis et au lacet que doit subir l'organe terminal du robot pour parvenir à la situation décrite. Cependant, une telle représentation comporte un énorme désavantage : il est quasi impossible de l'exploiter de manière mathématique (par exemple tester si l'axe des X d'un repère est parallèle ou perpendiculaire à l'axe des Y d'un autre repère, ...). C'est pourquoi une seconde représentation des voxelors a été définie de telle manière qu'elle soit plus exploitable

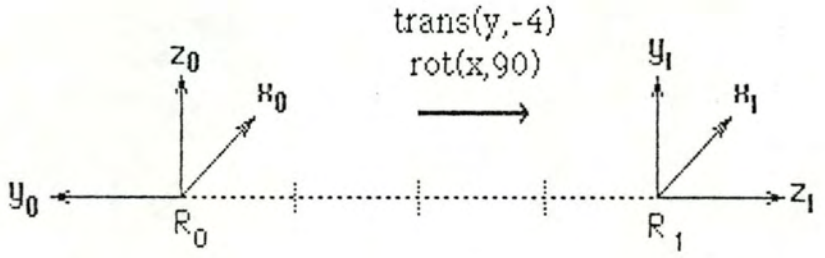
mathématiquement, quitte à s'éloigner d'une représentation qui serait pus proche de la couche "XYZ".

4.2.1.2. Voxelors sous forme matricielle.

Le modèle utilisé est basé sur celui développé par Paul [PAU81]. Chaque voxelor y est représenté par une matrice carrée 4x4 qui définit sa situation. Les trois premières colonnes représentent les coordonnées des vecteurs X, Y et Z du nouveau repère exprimé en fonction de R_0, tandis que la dernière colonne exprime les 3 translations du repère en fonction de R_0. La dernière ligne est constituée de trois 0 et de un 1 pour rendre la matrice carrée. L'orientation du voxelor est donc représentée par la sous-matrice 3x3 constituée des 3 premièrs éléments des 3 premières colonnes de la matrice 4x4, nous appelerons cette sous-matrice matrice orientation; tandis que la position du voxelor sera représentée par les 3 premiers éléments de la derniere colonne de la matrice que nous appellerons le vecteur position.



Exemple :



$$X_1 = 1\,X_0 + 0\,Y_0 + 0\,Z_0.$$

$$Y_1 = 0\,X_0 + 0\,Y_0 + 1\,Z_0.$$

$$Z_1 = 0\,X_0 - 1\,Y_0 + 0\,Z_0.$$

et

$$Tx = 0,\ Ty = -4,\ Tz = 0.$$

Ce qui donne $M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Remarques:

1. Une telle représentation est très facile à exploiter sur le plan mathématique. En effet, si l'on veut par exemple voir si l'axe des X d'un repère est paralèlle à l'axe des Z d'un autre repère, il suffit de voir si les éléments de la première colonne de la matrice orientation du premier repère sont identiques (au signe près, qui détermine le sens du vecteur) aux éléments de la troisième colonne de la matrice orientation du second repère.

2. Il semblerait qu'une telle représentation soit plus coûteuse à gérer d'un point de vue place qu'une représentation de type TCP (16 valeurs pour la représentation matricielle, 6 pour la rotationnelle). Or, il est possible de réduire le nombre de valeurs composant la matrice car celle-çi transporte des informations inutiles et redondantes:
   - Les 4 valeurs de la dernière ligne (0 0 0 1) qui servent à rendre la matrice carrée.
   - Une des colonne de la matrice orientation peut être déduite des 2 autres étant donné que la base est ortho-normée (le produit des deux premiers vecteurs forme le troisième).

   Il reste donc 9 valeurs utiles et nécessaires. Toutefois, pour une question de facilité, de lisibilité et de compréhension, nous n'utiliserons dans ce travail que la notation matricielle sous forme étendue. De même, pour avoir une traduction plus simple et plus

immédiate, les programmes travailleront avec la représentation sous forme étendue.

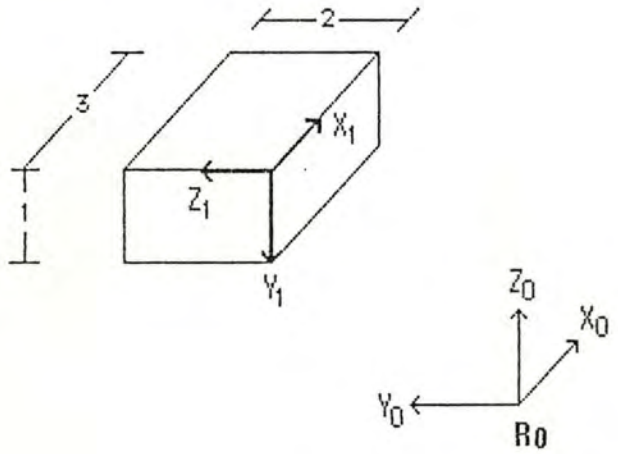
4.2.2. LES OBJETS PRIMITIFS.

Un objet primitif est donc une représentation compilée d'un ensemble de voxelors de même orientation.

Nous n'avons pris en considération dans ce travail que les parallélépipèdes rectanges, mais il est assez simple d'étendre les processus qui vont être développés à d'autres volumes symétriques (cylindres, sphères, ...).

Chaque objet primitif sera représenté par un voxelor (qui définira sa position et son orientation) que l'on appellera repère de l'objet primitif, et par un ensemble de paramètres qui définiront les dimensions de l'objet. Dans notre cas, un parallélépipède sera représenté par une matrice 4x4 définissant le repère du parallélépipède (un des coins au choix) et un vecteur de 3 paramètres donnant la longueur, la largeur et la hauteur (c'est-à-dire les trois dimensions suivant les trois axes du repère).

Exemple : soit l'objet primitif suivant :



Cet objet sera représenté par la matrice :

$$M = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 0 & -1 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

et le vecteur de paramètres
V = (3,1,2).

Notons qu'un autre coin aurait pu servir de repère à l'objet primitif; il aurait alors simplement donné lieu à une matrice différente et à une autre combinaison dans l'ordre des paramètres.

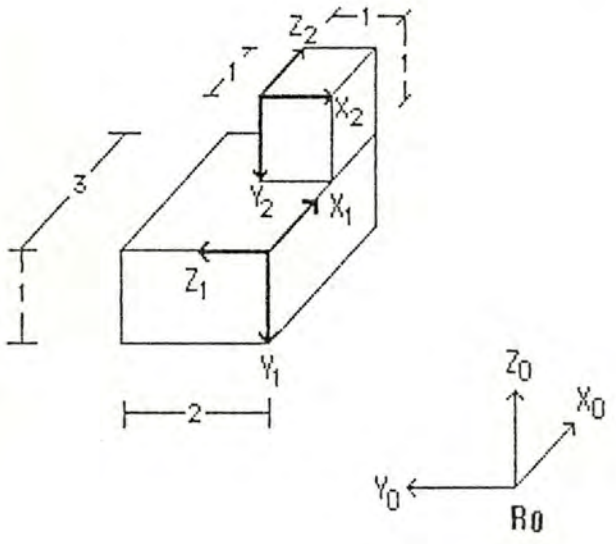
4.2.3. LES OBJETS.

Comme nous l'avons vu, un objet sera représenté par une composition d'objets primitifs suivant le modèle CGS-rep. Chaque objet sera

représenté par un voxelor (que nous appellerons repère principal), et par un ensemble de représentations d'objets primitifs (repère de l'objet primitif + paramètres). Le repère principal sera exprimé en fonction du repère absolu (R_0) ,tandis que les repères des objets primitifs seront exprimés en fonction du repère principal de l'objet. Pour simplifier la représentation, on fera coïncider le repère principal de l'objet avec le repère d'un des objets primitifs au choix.

Notons que, dans notre cas, un objet ne pourra être représenté que par l'union d'un ensemble de parallélépipèdes puisque nous n'avons considéré que l'union comme opérateur ensembliste et que les parallélépipèdes constituent les seuls objets primitifs que nous avons implémentés.

Exemple: soit l'objet suivant :



Cet objet sera représenté par :

- Le repère principal qui coïncide par exemple avec celui du parallélépipèdes P1 :

$$M = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 0 & 1 & 2 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Et la description des 2 parallélépipèdes dans le repère principal :

$$P1 : \quad R1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{et} \quad V = (3,1,2).$$

$$P2 : \quad R2 = \begin{pmatrix} 0 & 0 & 1 & 2 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{et} \quad V = (1,1,1).$$

Avantages de cette représentation:

1. Lors du déplacement d'un objet, il suffit de mettre à jour la situation du voxelor représentant le repère principal de l'objet, puisque la description relative des autres éléments par rapport à ce repère principal ne change pas.

2. L'erreur provoquée lors de la détermination de la situation absolue d'un voxelor quelconque de l'objet ne proviendra que d'une seule

erreur d'approximation puisque que chaque voxelor y est exprimé relativement au repère principal de l'objet. Il n'en serait pas de même s'ils étaient tous exprimés en absolu. En effet, comme nous l'avons deja souligné, le fait de travailler dans un monde discret provoque des situations non représentables lors du déplacement de voxelors dépendant les uns des autres. Nous serions dès lors forcés d'approximer la situation de ces voxelors à chaque déplacement d'objet et les erreurs d'approximation s'accumuleraient.

4.4.4. LES OBJETS COMPLEXES.

Les objets complexes sont donc des objets qui comportent des contraintes liées à leur seule existence ou à leur emploi ou encore à leur dépendance vis-à-vis d'autres objets. Ce niveau de la hiérarchie n'a pas été implémentée. Une façon possible de représenter les contraintes énumérées au point 4.1.5 serait par exemple pour chaque objet représenté :

- une liste des contraintes agissant sur cet objet seul.
- une liste de couple (objet, type de contrainte) definissant les contraintes provoquées par une relation entre l'objet considéré et l'objet défini dans le couple.

4.4.5. LES SITES.

Comme les sites sont des points particuliers de l'espace de travail du robot, ils seront représentés par un voxelor, c'est-a-dire par une matrice 4x4.

4.4.6. LES TUBES ET LES CIRCUITS.

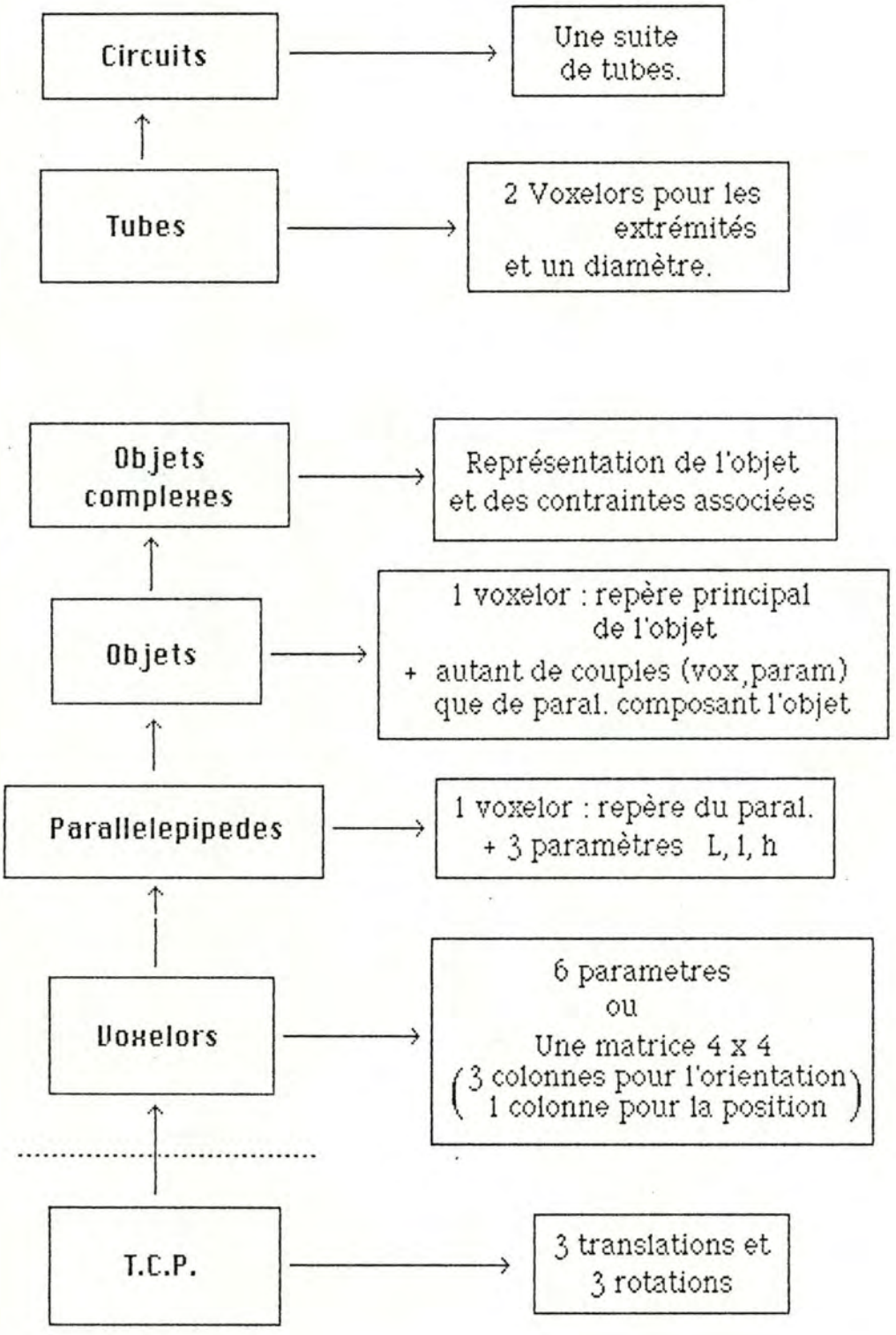
Un tube sera représenté par 2 voxelors, c'est-à-dire par 2 matrices 4x4 définissant les positions du TCP de la pince au départ et à l'arrivée du mouvement ainsi que par un paramètre définissant le diamètre du tube.

On gardera, pour définir les circuits, la représentation par une suite de tubes mis bout à bout.

4.4.7. REPRESENTATION GLOBALE DU MODELE.

Notons d'abord que la représentation choisie s'accorde très bien avec le désir d'utilisation immédiate formulé en début de chapitre. En effet, comme on peut le remarquer à la première page des annexes - qui donne la représentation en concept prolog du modèle - la traduction de la représentation du modèle en représentation par des concepts prolog est quasi immédiate.

Un résumé de la représentation des différentes composantes du modèle donne le schéma de la page suivante.


Circuits
Une suite de tubes.
Tubes
2 Voxelors pour les extrémités et un diamètre.
Objets complexes
Représentation de l'objet et des contraintes associées
Objets
1 voxelor : repère principal de l'objet
+ autant de couples (vox,param) que de paral. composant l'objet
Parallelepipedes
1 voxelor : repère du paral. + 3 paramètres L, l, h
Voxelors
6 parametres
ou
Une matrice 4 x 4
(3 colonnes pour l'orientation
1 colonne pour la position)
T.C.P.
3 translations et 3 rotations

5. ACTIONS SUR LES COMPOSANTES DU MODELE.

5.1. INTRODUCTION.

Brady [Bra85] prétend que le niveau objet de la programmation robotique fait partie intégrante de ce que l'on appelle Intelligence Artificielle. En effet, comme nous l'avons vu, la particularité de la programmation de niveau objet est de pouvoir spécifier le travail à accomplir sans plus devoir spécifier comment l'accomplir. La façon dont il faut accomplir le travail doit donc être générée automatiquement, et c'est précisément cette partie de la programmation que l'on classe sous l'étiquette "Intelligence Artificielle". L'I.A. intervient plus particulierement dans 2 domaines liés à la programmation objet : le "Grasping" (c'est-à-dire la saisie automatique d'objets) et le "path finding" (c'est-à-dire la recherche de trajectoire sans collisions lors du déplacement de la pince dans son espace de travail). Ces 2 sujets font l'objet de recherches très spécialisées qui sont loin d'être terminées.

Selon les objectifs que nous avons évoqués au début du chapitre 4, notre but dans ce chapitre sera de greffer une série de primitives à chacun des niveaux de la hiérarchie qui à été établie, chacune de ces primitives se situant au niveau considéré en utilisant des primitives de plus bas niveau. Plutôt que de nous disperser dans la recherche de primitives de bas niveau, nous avons decidé de nous concentrer sur les 2 problemes évoqués plus haut et plus particulièrement sur la saisie automatique. Nous

effectuerons donc un parcours assez étroit de bas en haut dans l'éventail des primitives qui pourraient exister en ne considérant que celles dont nous aurons besoin dans les niveaux supérieurs.

5.2. ACTIONS SUR LES VOXELORS.

5.2.1. TRANSFORMATION DE REPRESENTATION.

Comme nous l'avons vu dans la présentation du modèle, il est indispensable de pouvoir représenter un voxelor sous une forme matricielle ainsi que sous une forme rotationnelle, et de pouvoir passer indistinctement d'une représentation à l'autre. Rappelons qu'un voxelor sous forme rotationnelle s'exprime grâce à 6 paramètres représentant les 3 translations et les 3 rotations que le repère absolu doit subir pour parvenir a la situation souhaitée, tandis qu'un voxelor représenté au moyen de sa forme matricielle s'exprime grâce à une matrice 4x4 dont les 3 premières colonnes donnent les coéfficients directeurs des 3 axes du repère du voxelor en fonction du repère absolu; la dernière colonne contient les 3 translations à effectuer. Les 2 expressions qui suivent donnent les deux représentations d'un même voxelor.

$$(\,90°\,,\,180°\,,\,0°\,,\,-3\,,\,1\,,\,2\,) \;\equiv\; \begin{pmatrix} 1 & 0 & 0 & -3 \\ 0 & 0 & -1 & 1 \\ 0 & -1 & 0 & 2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

On remarque immédiatement que les 3 translations sont les mêmes. La conversion d'une représentation à l'autre est donc triviale au

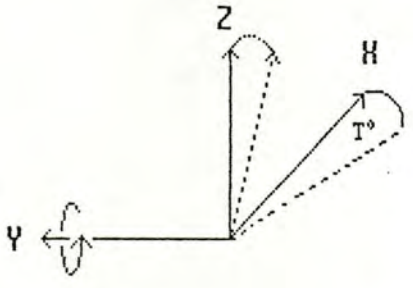
niveau des translations. Malheureusement, la conversion au niveau des rotations est beaucoup plus délicate.

1. De rotationnelle vers matricielle.

Nous savons que si le repère R_0 est exprimé par la matrice :

$$M0 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

la nouvelle base R1 obtenue a partir de R0 et d' une rotation de T° autour de l'axe de Y0 (tangage de T°) s'exprimera grâce à la matrice de transformation suivante :



$$rot(Y,T) = \begin{pmatrix} \cos T & 0 & \sin T \\ 0 & 1 & 0 \\ -\sin T & 0 & \cos T \end{pmatrix}$$

Et puisque chaque vecteur directeur de la nouvelle base est exprimé en fonction de la base de départ, nous pouvons écrire :

$$M1_{(R0)} = M0 \times rot(Y,T).$$

Ensuite, si nous faisons subir à cette nouvelle base un roulis de R°, nous avons :

$$M2_{(R0)} = M1_{(R0)} \times rot(X,R).$$

Enfin, si nous faisons subir à cette dernière base un lacet de L°, on obtient :

$$M3_{(R0)} = M2_{(R0)} \times rot(Z,L).$$

Une composition de ces 3 rotations successives se traduira donc par :

$$M3_{(R0)} = M0 \times rot(Y,T) \times rot(X,R) \times rot(Z,L).$$

Soit Tr la matrice ainsi obtenue:

$$Tr = \begin{pmatrix} ax & ox & nx \\ ay & oy & ny \\ az & oz & nz \end{pmatrix}$$

où ax = (cosT * cosR) + (sinT * sinR * sinL).
ay = (cosR * cosL).
az = (-sinT * sinL) + (cosT * sinR * sinL).

ox = (-cosT * sinL) + (sinT * sinR * cosL).
oy = (cosR * cosL).
oz = (sinT * sinL) + (cosT * sinR * cosL).

nx = (sinT * cosR).
ny = (-sinR).
nz = (cosT * cosR).

Nous pouvons donc assez facilement calculer la matrice de transformation Tr à partir des 3 angles T,R,L en appliquant simplement les formules décrites ci-dessus.

2. De matricielle vers rotationnelle.

L'opération inverse (calculer les 3 angles à partir de la matrice) est plus compliquée. En effet, d'après les formules ci-dessus, nous pouvons dire :

$$R = - \arcsin ny$$
$$T = \arccos (nz / \cos R)$$
$$L = \arccos (oy / \cos R).$$

Or, ces formules ne sont pas suffisantes pour deux raisons :

- La fonction arcsin n'est pas univoque ($\sin R = \sin(R+180°)$).
- Les deux dernières équations sont indéterminées pour R = 90° ou 270°.

Il nous faut donc aborder le problème par une autre approche.

Cette autre approche va demander l'utilisation d'une fonction trigonométrique particulière : la fonction arctg2. Cette fonction nous sera utile car elle a la propriété d'être univoque. On peut la définir comme suit :

Pour tout couple X,Y désignant les coordonnées d'un point inscrit sur la circonférence d'un cercle de rayon 1 et dont le centre constitue la base ortho-normée dans laquelle sont exprimés X et Y, la fonction arctg2(X,Y) renvoie un angle - compris entre -180° et +180°-, formé par le vecteur de coordonée (X,Y) et l'axe horizontal.

Autrement dit, arctg2(X,Y) = arctg(Y/X) mais en tenant compte du signe des 2 arguments pour restituer un résultat unique.

Reconsidérons la matrice de transformation que nous avions obtenue par rotation successive de la base autour de ses 3 axes:

$$Tr = rot(Y,T) \times rot(X,R) \times rot(Z,L) .$$

Et donc :

$$(rot(Y,T))^{-1} \times Tr = rot(X,R) \times rot(Z,L).$$

C'est-à-dire :

$$\begin{pmatrix} cosT & 0 & -sinT \\ 0 & 1 & 0 \\ sinT & 0 & cosT \end{pmatrix} \begin{pmatrix} ax & ox & nx \\ ay & oy & ny \\ az & oz & nz \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cosR & -sinR \\ 0 & sinR & cosR \end{pmatrix} \begin{pmatrix} cosL & -sinL & 0 \\ sinL & cosL & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

d'où :

$$\begin{pmatrix} f11(a) & f11(o) & f11(n) \\ f12(a) & f12(o) & f12(n) \\ f13(a) & f13(o) & f13(n) \end{pmatrix} = \begin{pmatrix} cosL & -sinL & 0 \\ cosR*sinL & cosR*cosL & -sinR \\ sinR*sinL & sinR*cosL & cosR \end{pmatrix}$$

où l'on a posé : f11 = cosT*x - sinT*z
f12 = y
f13 = sinT*x + cosT*z

p. ex : f12(o) = oy
f13(n) = sinT*ny + cosT*nz

On remarque :

f11(n) = 0 ==> cosT*nx - sinT*nz = 0
==> sinT/cosT = nx/nz

==> **T = arctg2(nx,nz)**

Si nx et nz sont tous les deux nuls, alors T est indéfini. Cela signifie que n'importe quel angle peut convenir et qu'il sera toujours compensé par

les deux autres rotations pour faire parvenir le repère à l'orientation voulue. Nous le fixerons arbitrairement à 0.

Par ailleurs, on a

f12(n) = -sinR	d'où	-sinR = ny
f13(n) = cosR		cos R = sinT*nx + cosT*nz

==> **R = arctg2(-ny , sint*nx + cosT*nz).**

Et on a également

f11(a) = cosL	d'où	cosL = cosT*ax + sinT*az
f11(o) = -sinL		-sinL = cosT*ox - sinT*oz

==> **L = arctg2(-cosT*ox + sinT*oz , cosT*ax - sinT*az)**

5.2.2. CHANGEMENT DE BASE.

Lorsque le repère d'un voxelor est exprimé en fonction du repère d'un deuxième voxelor, lui-même exprimé en fonction d'un troisième repère, il est assez aisé d'exprimer le repère du premier en fonction du troisième grâce à la représentation matricielle. En effet, Paul [PAU81] écrit au sujet des matrices de transformation que : " Si l'on post-multiplie une transformation représentant une base par une seconde transformation décrivant une rotation, alors on fait cette rotation en respect des axes décrits dans la première transformation."

D'où l'on peut écrire : $M2_{(R0)} = M1_{(R0)} \cdot M2_{(R1)}$

Ce qui est en fait une simple multiplication de 2 matrices (4 x 4).

5.3. ACTIONS SUR LES OBJETS PRIMITIFS.

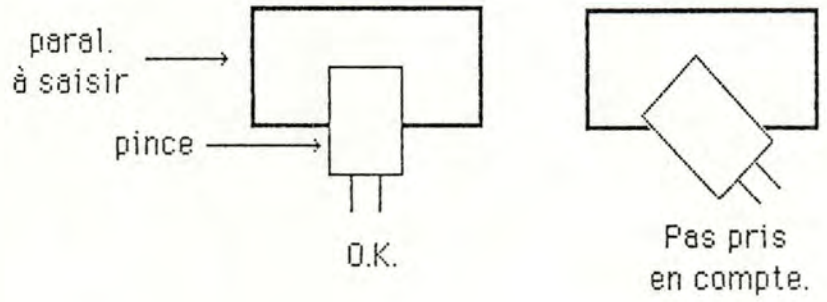
5.3.1. RECHERCHE DES POINTS DE SAISIE D'UN PARALLELEPIPEDE.

Un point de saisie d'un objet est un voxelor appartenant à l'objet et définissant une situation telle que si on y amène le TCP de la pince, on puisse saisir l'objet. Pour la saisie automatique d'objets, il serait intéressant de disposer de l'ensemble des points de saisie de l'objet que l'on veut prendre afin de pouvoir en choisir un. Nous allons donc essayer de trouver un procédé qui détermine automatiquement l'ensemble des points de saisie d'un objet d'après sa configuration géométrique. Les primitives de ce niveau détermineront les points de saisie d'un objet primitif et dans notre cas d'un parallélépipède.

Notons que pour saisir un parallélépipède, il faut qu'au moins une de ses dimensions (suivant l'axe des X, des Y ou des Z de son repère) soit plus étroite que la largeur d'écartement maximun de la pince. Considérons pour l'instant un parallélépipède qui n'ait qu'une seule dimension qui satisfait cette condition. (par exemple suivant X).

Pour simplifier les calculs et pour être sûr que la pince ait une bonne prise, on va ne prendre en compte que les points de saisie tels que, si la pince s'y positionne, le plan formé par les axes Y,Z du TCP de la pince soit parallèle à l'axe des X du repère du parallélépipède.

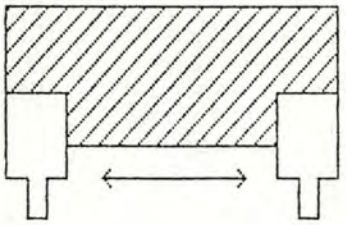
Vue de haut.



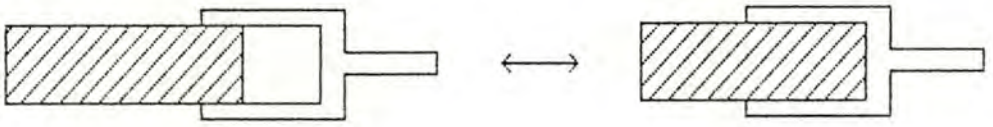
Donc, dans le cas considéré il y aura 8 orientations possibles pour le TCP de la pince : 2 pour chaque côté dont la dimension satisfait la condition (comme la pince est symétrique, si on lui fait subir un roulis de 180°, elle prendra une position qui paraît identique à celle précédant le roulis bien qu'elle ait fait un demi-tour sur elle-même).

Pour chacune des orientation que peut prendre le TCP de la pince, examinons les positions qu'il peut prendre en effectuant des translations sur les 3 axes de son repère :

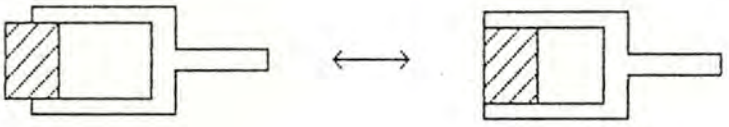
- sur l'axe des Z : on pourra faire glisser la pince le long du parallélépipède à saisir; nous prendrons comme convention que la pince pourra glisser tout le long du parallélépipède à saisir mais qu'elle ne pourra pas dépasser sur les côtés.



- sur l'axe des X : on pourra enfoncer plus ou moins profondément la pince dans l'objet; nous prendrons comme convention qu'elle devra toujours être enfoncée à au moins un tiers de sa longueur si le parallélépipède qu'elle veut saisir est plus long que la pince.



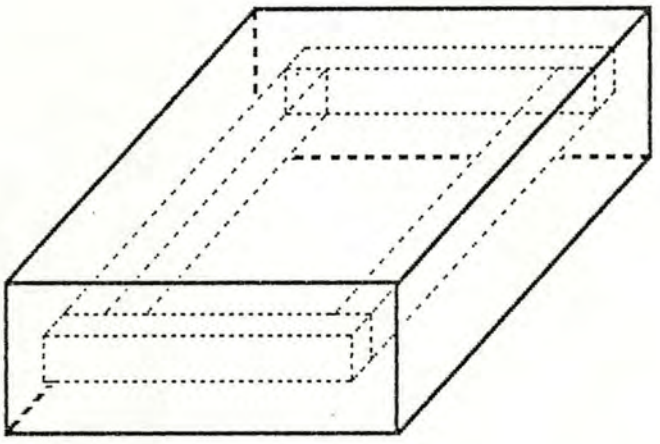
et qu'elle devra toujours être enfoncée au minimum jusqu'a la moitié du parallélépipède et au maximum jusqu'au bout du parallélépipède si ce dernier est plus petit que la longueur de la pince.



- sur l'axe des Y : aucune translation n'est permise car le parallélépipede doit être centré par rapport aux deux machoires de la pince, sinon quand celles-ci se refermerons pour saisir l'objet, le parallélépipede risquerait de bouger.

Par ces 3 translations nous avons en fait défini un nouveau parallélepipede qui sera à l'intérieur du parallélépipède à saisir et dont les dimensions seront : la largeur du parallélépipède à saisir moins la largeur de la pince car le TCP est situé au milieu, 2/3 de la longueur de la pince ou la moitié de la longueur du parallélépipède à saisir, et 1 pour la dernière

dimension. Nous nommerons un tel parallélépipède parallélépipède de saisie.



A chaque orientation possible du TCP de la pince sera donc associé un ensemble de positions, le tout formant un parallélépipède de saisie. Dans notre cas, nous aurons donc 8 parallélépipèdes de saisie possibles (on n'en voit que 4 car les 4 autres leur sont juxtaposés puisque la pince est symétrique) Remarquons que si une deuxième dimension du parallélépipède à saisir avait satisfait à la condition d'être plus petite que la distance d'écartement maximum de la pince, ce parallélépipéde aurait contenu 16 parallélépipèdes de saisie; tandis que si les 3 dimensions étaient satisfaisantes il y en aurait eu 24.

Maintenant que nous avons senti intuitivement la manière de procéder pour déterminer les parallélépipèdes de saisie d'un parallélépipède à saisir, nous allons formaliser notre raisonnement. Dans un premier temps, nous allons chercher une façon de générer toutes les orientations possibles que peut prendre la pince pour saisir un parallélépipède (comme nous

l'avons vu, il y en aura toujours 0,8,16 ou 24). Ces orientations détermineront les orientations des repères des parallélépipèdes de saisie. Ensuite, pour chacune des orientations générées, nous devrons déterminer une position pour le repère parallélépipède de saisie. Enfin, dans le même temps, nous déterminerons les dimensions de ce parallélépipède suivant chacun des 3 axes de son repère.

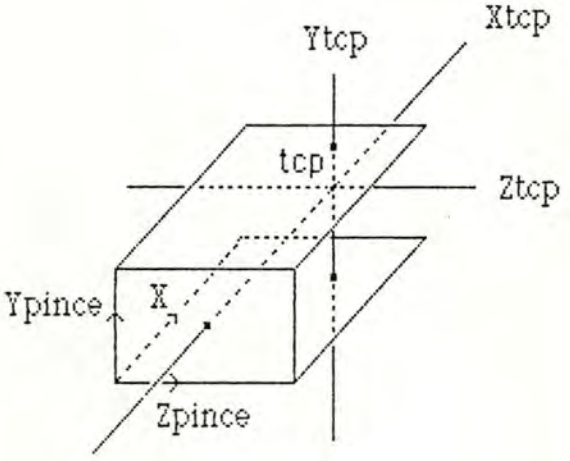
1. Recherche des orientations de la pince.

Notons d'abord que la pince peut également se représenter par un parallélépipède ("ouvert" car il manque 3 faces) où :

Lxpince = longueur de la pince,

Lypince = distance d'écartement maximum.

Lzpince = largeur de la pince.

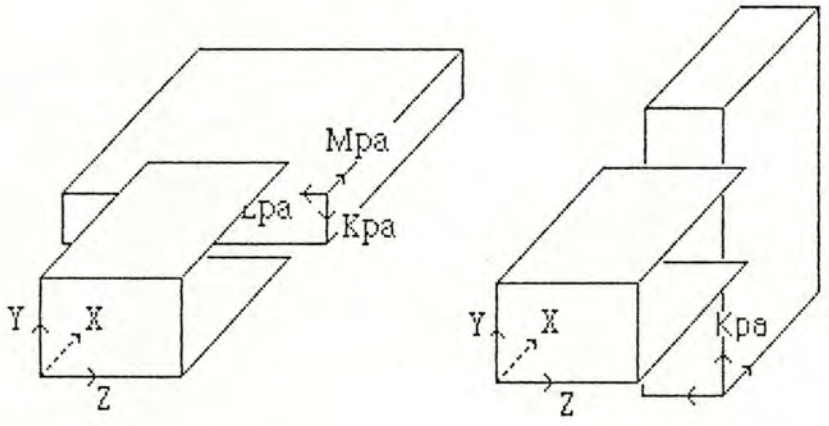


Nous fixerons le repère de ce parallélépipède au seul coin possible du parallélépipède qui lui permette de garder la même orientation que celle du

TCP de la pince (dans le coin inférieur gauche et devant sur le schéma). Tout en ayant des valeurs positives pour ses dimensions suivant les 3 axes (sinon, on aurait pu, par exemple, placer le repère au coin avant supérieur gauche en gardant la même orientation et en spécifiant une longueur négative suivant l'axe des Y).

Sa représentation dans le modèle sera une matrice accompagnée d'un vecteur de 3 paramètres : Lxpince,Lypince,Lzpince.

Pour chacun des axes du parallépipède de départ, si la longueur du parallélépipède suivant cet axe est plus petite que la largeur d'écartement de la pince (Lypince), nous séléctionnerons cet axe. Soit Kpa l'axe séléctionné (Kpa = Xparal, Yparal ou Zparal). A chacun des axes séléctionnés, nous allons faire subir l'ensemble du traitement décrit ci-dessous



On remarque que pour décrire une des 8 orientations que peut prendre la pince pour saisir le parallélépipède suivant l'axe Kpa séléctionné :

- l'axe Y du repère de la pince doit être parallèle à Kpa en étant soit de même sens,soit de sens contraire. (Y vaudra donc Kpa ou -Kpa)

- l'axe X du repère de la pince doit être parallèle à n'importe lequel des deux axes Lpa et Mpa non séléctionnés du repère du parallélépipède à saisir en ayant indifférement le même sens ou le sens contraire. (X sera donc équivalent à un des vecteurs suivant : Lpa, -Lpa, Mpa, -Mpa).

- L'axe Z du repère de la pince se déduit des 2 premiers : puisqu'ils forment à eux trois une base orthogonale, le produit vectoriel des 2 premiers vecteurs forme le troisième : Z = X x Y.

Les 8 solutions seront données par la combinaison des différentes possibilités que nous avons établies :

(Kpa, Lpa, Kpa x Lpa)	(-Kpa, Lpa, -Kpa x Lpa)
(Kpa, -Lpa, Kpa x -Lpa)	(-Kpa, -Lpa, -Kpa x -Lpa)
(Kpa, Mpa, Kpa x Mpa)	(-Kpa, Mpa, -Kpa x Mpa)
(Kpa, -Mpa, Kpa x -Mpa)	(-Kpa, -Mpa, -Kpa x -Mpa)

Chaque matrice orientation ainsi obtenue exprime donc l'orientation de la pince en fonction du repère du parallélépipède à saisir.

2. Arguments du parallélépipède de saisie.

Il nous faut maintenant déterminer les dimensions du parallélépipède de saisie suivant les 3 axes qui déterminent son repère, ceci pour chaque matrice orientation.

Soit (X,Y,Z) une des matrices d'orientation.

Nous cherchons :

Lxp, Lyp, Lzp les 3 paramètres délimitant les dimensions du parallélépipède de saisie; d'après les conventions et les considérations faites précédemment, nous pouvons écrire :

Lxp = (2/3 de Lxpince) si le parallélépipède à saisir est plus long que la pince.
= (longueur du parallélepipède à saisir suivant l'axe parallèlè à Xpince) / 2
si le parallélépipède est plus court que la pince.

Lzp = (longueur du paralleélépipède à saisir suivant l'axe parallele à Zpince - Lzpince). Il faut soustraire la largeur de la pince puisque le TCP se trouve au milieu de Zpince.

Lyp = toujours 1.

3. Position du repère du parallélépipède de saisie.

Nous devons également, pour chaque matrice orientation, déterminer la position du repère du parallélépipède de saisie, c'est-à-dire les 3 translations que doit subir le repère du parallélépipède de saisie par rapport au repère du parallélépipède à saisir. Ici aussi, nous fixerons la position du repère du parallélépipède de saisie au seul coin possible du parallélépipède tel que toutes ses dimensions soient exprimées positivement. Les repères des parallélépipèdes de saisie seront donc toujours au même

coin par rapport à l'orientation du TCP de la pince qui est en juxtaposition avec un des voxelors du parallélépipède à saisir.

Nous nommerons Dk, la translation suivant l'axe des K du repère du parallélépipède de départ.

soit Kpa un des 3 axes du parallélépipède à saisir :

Si c'est X qui lui est parallèle :

Si Lk > Lxpince
Dk = 1/3 de Lxpince si Kpa et X ont le même sens.
Dk = Lk - 1/3 Lxpince s'ils sont de sens contraire.

Si Lk< Lxpince
Dk = Lk / 2 quel que soit le sens.

Si c'est Y qui lui est parallele :

Dk = Lk / 2 quel que soit le sens.

Si c'est Z qui lui est paralelle :

Dk = 1/2 Lzpince si Kpa et Z ont le même sens.
Dk = Lk - 1/2 Lzpince s'ils sont de sens contraire.

Nous avons donc à ce stade une série de matrices (4 x 4) indiquant chacune l'orientation et la position possible du repère d'un parallélépipède de saisie ainsi qu'une série de paramètres précisant les dimensions possibles de chacun d'eux. Il reste donc à transformer ces matrices pour qu'elles expriment ces orientations et positions en fonction de R_0 plutôt qu'en fonction du repère du paral. Pour cela, il suffit d'appliquer la formule de Paul vue précédemment et donc d'utiliser la primitive de changement de base :

$$Tr_{(R0)} = Repère\ paral_{(R0)} . Tr_{(Repère\ paral)}$$

5.3.2. RECOUVREMENT DE DEUX PARALLELEPIPEDES.

Nous dirons que deux objets se recouvrent si l'intersection des volumes des 2 objets n'est pas vide. Si cette intersection est vide, nous dirons que les 2 objets sont disjoints. Comme nous le verrons, cette primitive nous sera très utile, aussi bien pour déterminer les parallélépipèdes de saisie d'objets que dans la recherche de trajectoires sans collisions.

La méthode employée permet une utilisation optimale de la représentation matricielle des repères des parallélépipèdes. En effet, nous nous sommes basés sur le fait suivant :

Nous dirons qu'un ensemble de points se trouvent du même côté de 2 plans parallèles P1 et P2 si tous ces points sont du même côté de P1 et en même temps, du même côté de P2, sans se trouver dans l'espace situé entre les 2 plans.

Considérons les 3 couples de faces parallèles de chaque parallélépipède. Les parallélépipèdes sont disjoints s'il existe au moins un couple de faces appartenant à l'un ou l'autre parallélépipède tel que tous les coins de l'autre parallélépipède se trouvent du même côté des 2 plans définis par les faces de ce couple. Si un tel couple n'existe pas, alors les parallélépipèdes sont imbriqués.

Pour optimaliser la méthode, nous utiliserons préalablement la condition suffisante suivante pour que deux parallélépipèdes soient disjoints (si elle est satisfaite, on est sûr qu'ils sont disjoints, sinon on ne peut rien dire) :

Si R1 , LXp1 , LYp1 , LZp1
et R2 , LXp2 , LYp2 , LZp2

sont respectivement les 2 repères et les dimensions des 2 parallélépipèdes,

si Max(LXp1,LYp1,LZp1) + Max(LXp2,LYp2,LZp2) < Distance(R1,R2)

alors P1 et P2 sont disjoints.

Nous devons donc, avec la représentation matricielle :

1. Calculer la distance entre les 2 repères.

   Soient (TX1,TY1,TZ1,1) et (TX2,TY2,TZ2,1)

   les dernières colonnes des matrices représentant R1 et R2
   (Ces colonnes contiennent donc les translations suivant les
   3 axes du repère absolu qu'ont dû subir R1 et R2
   pour parvenir à leur position actuelle).

   Alors:

   $$Distance(R1,R2) = ((TX1 - TX2)^2 + (TY1 - TY2)^2 + (TZ1 - TZ2)^2)^{1/2}$$

2. Trouver les coéfficients des équations des plans contenant les faces des parallélépipèdes considérés, ce qui est assez aisé grâce à la représentation matricielle. En effet, nous disposons du repère sous une forme exprimant les vecteurs directeurs de chacun des axes. Il suffit dès lors de considérer ces vecteurs 2 à 2 pour qu'ils définissent chacun un plan comprenant une face du parallélépipède. Les équations des trois plans restants découlent ensuite facilement de celles des 3 premiers puisqu'il suffit de faire subir à chacun d'eux une translation d'une longueur égale à la dimension du

parallélépipède suivant le vecteur qui n'a pas servi à la construction du plan.

3. Déterminer la position des 8 points de chaque parallélépipède. Cette opération se realise également assez facilement grâce à la représentation matricielle. En effet, la position de chacun de ces points en fonction du repère du parallélépipède auquel ils appartiennent est chaque fois triviale puisqu'elle résulte de 0, 1, 2, ou 3 translations suivant les 3 axes du repère du parallélépipède; l'amplitude de ces translations sera égale à la longueur du parallélépipède sur l'axe considéré. Il suffit alors d'introduire les coordonnées relatives de ces points dans l'équation de Paul pour connaître leur position par rapport à R_0.

4. Déterminer si les 8 coins d'un parallélépipède sont tous du même côté que les 2 faces parallèles de l'autre parallélépipède. En effet, il suffit d'appliquer la coordonnée de tous les points aux 2 équations des plans contenant les faces considérées et de verifier si les résultats des équations ont bien tous le même signe.

5.3.3. MISE A JOUR DE LA POSITION D'UN PARALLELEPIPEDE.

Avec, en entrée, le nom du parallélépipède et la nouvelle situation de son repère, cette primitive va mettre à jour la base de donneés en remplacant le voxelor indiquant l'ancienne situation par un voxelor indiquant la nouvelle situation.

5.4. ACTIONS SUR LES OBJETS.

5.4.1. RECOUVREMENT D'OBJETS.

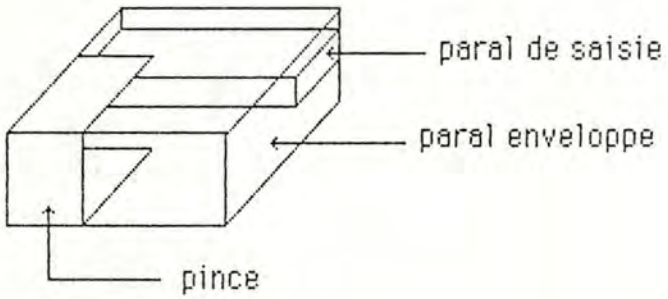
Pour déterminer si 2 objets sont disjoints ou non, il suffit de regarder si les parallélépipèdes composant le premier objet sont bien tous disjoints de tous les parallélépipèdes composant le deuxième objet.

5.4.2. RECHERCHE DES POINTS DE SAISIE D'UN OBJET.

Comme nous l'avons vu, les objets que nous avons considérés dans ce travail sont composés d'une série de parallélépipèdes. Nous savons déjà déterminer les parallélépipèdes de saisie d'un parallélépipède à saisir, mais il est évident que n'importe lequel d'entre eux ne peut convenir ici. En effet, 2 parallélépipèdes composant l'objet peuvent avoir une face commune par laquelle on ne pourra donc pas saisir l'objet. Pour éliminer les parallélépipèdes qui ne conviennent pas, nous allons employer la primitive de recouvrement.

Pour chacun des parallélépipèdes de saisie de chaque parallélépipède composant l'objet, nous allons créer un nouveau parallélépipède tel qu'il contienne la pince quelle que soit la position de cette dernière dans le parallélépipède de saisie. Nous appelerons un tel parallélépipède parallélépipède enveloppe. Il suffira alors de tester si ce parallélépipède enveloppe est bien disjoint de tous les parallélépipèdes primitifs composant l'objet (sauf bien sûr de celui avec lequel il a été construit). S'il est effectivement disjoint de tous les parallélépipèdes, cela signifie que la pince pourra se positionner à n'importe quel point du parallélépipède de saisie enveloppé sans provoquer de collisions avec les

autres parallélépipèdes composant l'objet . Dans le cas où l'on aurait trouvé un parallélépipède qui recouvrirait le parallélépipède enveloppe, cela signifierait que la juxtaposition du TCP de la pince à au moins un voxelor du parallélépipède de saisie provoquerait une collision entre la pince et ce parallélépipède primitif. Un tel parallélépipède de saisie devra être éliminé.



Construction du parallélépipède enveloppe :

Nous devons trouver la position et l'orientation du repère du parallélépipède enveloppe ainsi que les valeurs des 3 parametres mentionnant ses dimensions.

- L' orientation de ce parallélépipède enveloppe est triviale puisqu'elle est la même que celle du parallélépipède de saisie. (donc les 2 matrices orientation seront égales).

- La position de ce parallélépipède s'exprimera grâce à 3 translations suivant les 3 axes du repère du parallélépipède de saisie. Pour déterminer la longueur des ces 3 translations, il suffit d'envisager la pince dans une situation telle que son TCP coïncide avec le repère du parallélépipède de saisie et d'imaginer les 3 translations que devrait subir le TCP pour aller rejoindre le repère de la pince. Il devrait :

1. Reculer jusqu'au fond de la pince
   c'est-à-dire opérer une translation négative sur l'axe des X de longueur égale à la longueur de la pince.
   $T_x = -Lx_{pince}$.

2. Descendre au bord inférieur de la pince
   c'est-à-dire opérer une translation négative sur l'axe des Y de la moitié de la distance d'écartement maximale de la pince.
   $T_y = -Ly_{pince}/2$.

3. Se déplacer latéralement vers la gauche jusqu'à rencontrer le coin qui sert de repère au parallélépipède décrivant la pince
   c'est à dire opérer une translation négative sur l'axe des Z de la moitié de la largeur de la pince.
   $T_z = Lz_{pince}/2$

- Les dimensions du parallélépipède enveloppe peuvent s'obtenir de façon similaire. En effet, comme ce parallélépipède doit contenir la pince dont le TCP coïnciderait avec n'importe quel voxelor du parallélépipède de saisie; il suffit de s'imaginer le TCP de la pince parcourant tous les voxelors du parallélépipède de saisie et considérer les situations extrêmes :

1. Dimension suivant l'axe des X: déplacement maximum du du TCP suivant l'axe des X du parallélépipède de saisie (Lxp) plus la longueur de la pince.
   ($Lx_{pe} = Lx_p + Lx_{pince}$)

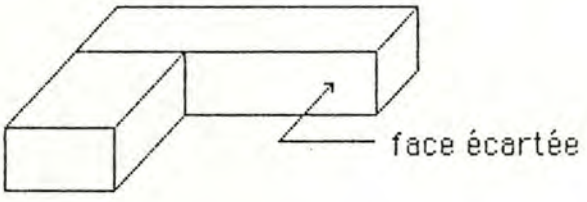
2. Dimension suivant l'axe des Y : comme le parallélépipède de saisie a une dimension égale à 1 suivant cet axe là, le TCP ne pourra pas bouger et seul l'écartement maximum de la pince déterminera la dimension.
   ($Ly_{pe} = Ly_{pince}$)

3. Dimension suivant l'axe des Z : déplacement maximum du TCP de la pince suivant l'axe des Z du parallélépipède de

saisie plus la longueur de la pince puisque son TCP se situe au milieu de la pince.
(Lzpe = lzp + Lzpince)

Pour éliminer également les saisies qui seraient impossibles de par la proximité d'un autre objet, nous pourrions également tester si ce parallélépipède enveloppe est disjoint des objets compris dans l'espace de travail (et donc pour éliminer aussi les faces inférieures d'objets posés à même la table, il suffira d'avoir représenté celle-ci par un parallélépipède dans la B.D.)

L'inconvénient de la méthode est que l'on va systématiquement écarter des faces qui seraient au moins partiellement utilisables.



5.4.3. MISE A JOUR DE LA POSITION D'UN OBJET.

Avec, en entrée, le nom de l'objet et la nouvelle situation de son repère principal, cette primitive va mettre à jour la base de donneés en remplaçant le voxelor indiquant l'ancienne situation par le voxelor indiquant la nouvelle situation. Ceci se fera en utilisant la primitive de mise à jour de la position d'un parallélépipède en mettant à jour la situation du

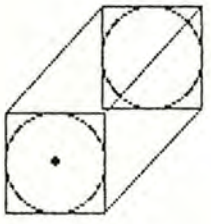
parallélépipède de l'objet contenant le repère principal (comme les repères des autres parallélépipèdes sont exprimés relativement à ce repère principal, ils seront automatiquement mis à jour également).

5.5. ACTIONS SUR LES TUBES.

5.5.1. CONSTRUCTION DE TUBES.

Nous disposons dans notre modèle de la représentation de tubes par un diamètre et par 2 voxelors définissant les extrémités. Les tubes serviront à définir des enveloppes cylindriques dans lesquelles passera la pince lors de ses déplacemenst. Or, un des concepts importants au niveau objet est celui de pouvoir définir des trajectoires sans collisions avec les objets du monde de travail du robot. Nous allons donc nous servir des tubes pour détecter si il y aura ou non une collision lors d'un déplacement en testant si il n'y a pas de recouvrement entre le tube dans lequel s'effectuera le déplacement et les objets definis dans la base de données des objets.

Comme nous disposons d'une méthode déterminant si 2 parallélépipèdes sont disjoints ou non, nous allons envelopper les tubes dans des parallélépipèdes :



Il suffira alors, lors d'un déplacement, de tester si ce parallèlépipède recouvre des objets de l'espace de travail pour savoir si le déplacement est valable.

Méthode de construction de tubes sous enveloppes parallélépipèdiques :

Soit un tube T défini par 2 voxelors R1 et R2 et par un diamètre D.
Nous recherchons un parallélépipède defini par un voxelor R et par ses 3 dimensions : Lx, Ly, Lz tel qu'il enveloppe le tube T.

1. Orientation du voxelor repère du parallélépipède (R).

   a. La direction du vecteur X est donnée par la position des 2 extrémités du tube :

      $ax' = (x2-x1)$; $bx' = (y2-y1)$; $cx' = (z2-z1)$.

      On divise alors chacun des 3 coéfficients par la somme des 3 pour normaliser la direction : $N = |x2-x1| + |y2-y1| + |z2-z1|$.

      $$ax = ax'/N \; ; \; bx = bx'/N \; ; \; cx = cx'/N.$$

      $$(\Longrightarrow ax + bx + cx = 1)$$

   b. La direction du vecteur Y est quelconque pourvu qu'il soit perpendiculaire au vecteur X

      $$\Longrightarrow (ax * ay) + (bx * by) + (cx * cy) = 0.$$

      (avec ay, by, cy non tous nuls).

On pose : ay = 0, by = 1 -> cy = -bx/cx si cx est différent de 0.

ay = 0, cy = 1 -> by = -cx/bx si bx est différent de 0.

cy = 1, by = 0 -> ay = -cx/ax si ax est différent de 0.

Et il reste alors à normaliser le vecteur.

c. La direction de Z dépend des 2 premiers puisque c'est une base orthogonale (produit de X par Y).

az = bx*cy - cx*by

bz = ax*cy - cy*ay

cz = ax*by - bx*ay

2. Position du voxelor repère du parallélépipède.

Comme le voxelor de départ est situé au centre du tube de diametre D, il faut lui faire subir une translation de (0 , -D/2 , -D/2), et donc, pour avoir la position du repère, il suffit d'appliquer la formule du changement de base de Paul : $R_{(R0)} = R1_{(R0)} . R_{(R1)}$

Ce qui donne:

$$\begin{pmatrix} ax' & ay' & az' & dx' \\ bx' & by' & bz' & dy' \\ cx' & cy' & cz' & dz' \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} ax & ay & az & dx \\ bx & by & bz & dy \\ cx & cy & cz & dz \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -D/2 \\ 0 & 0 & 1 & -D/2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Le fait que la seconde matrice soit proche de la matrice identité rend la multiplication matricielle plus légère. En effet, seule la dernière colonne de la matrice résultat change par rapport à R1. Et donc :

dx' = -(ay+az)*D/2 + dx

dy' = -(by+bz)*D/2 + dy

dz' = -(cy+cz)*D/2 + dz

3. Dimensions du parallélépipède.

Suivant l'axe de Y et de Z : D.

Suivant l'axe des X : distance entre les 2 extrémités du tube.

==> Lx = sqrt((dx2-dx1)**2 + (dy2-dy1)**2 + (dz2-dz1)**2).

5.5.2. DETERMINER SI UN TUBE EST SANS COLLISION.

Un tube est sans collision s'il ne recouvre aucun des objets présents dans l'espace de travail. Il suffira donc, pour déterminer s'il est ou non sans collision, de lui appliquer la primitive de recouvrement avec tous les objets présents dans la base de données.

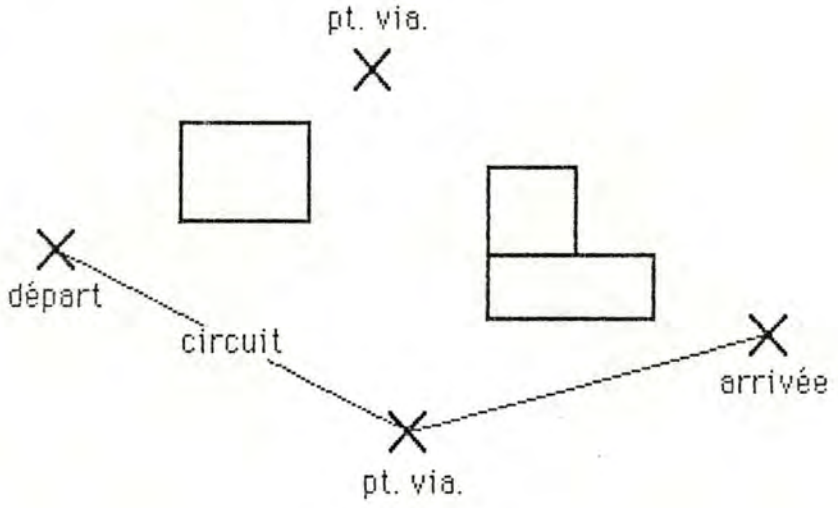
5.6. ACTIONS SUR LES CIRCUITS.

5.6.1. CONSTRUCTION D'UN CIRCUIT SANS COLLISION.

La recherche d'une trajectoire sans collision est réellement un problème très complexe si on veut rester performant. En effet, dès qu'on le simplifie un tant soit peu, on aboutit très vite à des problèmes sans solutions, même dans des environnements faiblement encombrés (en effet, la pince ne bouge plus car elle ne trouve aucune trajectoire sans collision). Comme nous n'avons pas la prétention de résoudre ce problème, nous proposons une solution simple qui ne fonctionne que s'il y a peu d'objets dans l'espace de

travail et qui demande une certaine aide extérieure. Il ne nous est de toute façon pas possible de résoudre complètement ce problème puisque nous n'avons plus conscience, à ce niveau de la programmation, de l'intégralité du bras articulé. Seul l'organe terminal est encore sous notre contrôle.

On considère que le programmeur a préalablement défini dans l'espace un certain nombre de points vias : ce sont des points par où pourra transiter la pince si un obstacle se trouve sur son chemin.



La primitive déterminera donc un ensemble de tubes mis bout à bout, qui constitueront un circuit, au départ de 2 voxelors à relier, d'une longueur définissant le diamètre des tubes constituant le circuit et d'un ensemble de points vias possibles. Notons que pour simplifier, tous les tubes auront un diamètre identique; il sera déterminé par la taille de l'objet plus la tolérance du tube de niveau "XYZ" . En effet, comme il faut construire un circuit sans collision, nous devons veiller à ce que les tubes définis dans ce circuit enveloppent tout point de l'espace dans lequel pourra se trouver

l'objet lors de son déplacement. On pourrait peut-être augmenter la performance en améliorant la technique suivante : les tolérances des tubes de niveau "XYZ" pourraient être définies automatiquement; pour relier 2 points vias, on définirait un tube dont le diamètre serait celui de la taille de l'objet à déplacer plus une tolérance minimale (et donc le trajet s'effectuerait en ligne droite mais avec une extrême lenteur). Si ce tube, aprés vérification, est déclaré sans collision, on pourrait le redéfinir avec une tolérance un peu plus grande de telle manière à accélérer le mouvement lorsqu'on le parcourra. On verifiera alors que ce tube est bien sans collision et s'il l'est on continuera à augmenter la tolérance j'usqu'à obtenir un tube de diamètre tel qu'une collision sera détectée. Le tube utilisé pour définir cette partie du circuit sera alors celui qui comportera la tolérance la plus élevée sans pour autant provoquer une collision.

Nous avons implémenté la construction de tubes sans collision en considérant un diamètre unique pour tous les tubes du circuit. Ce problème a été assez facilement résolu par la recherche d'un chemin dans un graphe.

5.7. ACTIONS SUR LES OBJETS ET LES TUBES.

5.7.1. DEPLACEMENT D'UN OBJET DANS UN TUBE.

La primitive se présentera avec, en entrée, un nom d'objet à déplacer, le nom de la pince qui déplace l'objet et un tube (2 voxelors et un diamètre), le nom de l'objet sera éventuellement vide si seule la pince est à déplacer. S'il est présent, l'objet aura préalablement été saisi par la pince. Notons que lorsque cette primitive sera activée, cela signifiera que la primitive de niveau supérieur qui l'aura appelée aura préalablement

determiné que le parcours entre les 2 voxelors donnés en entrée s'effectuera sans collision grâce à la primitive de détermination de tube sans collision. Le diamètre du tube demandé ici sera donc équivalent à la tolérance du tube de niveau "XYZ".

Il suffit, dès lors, de faire appel à la primitive de déplacement de la pince dans un tube du niveau "XYZ" et de mettre à jour la position des objets déplacés dans la base de données.

Pour réaliser la mise à jour de l'objet déplacé, il faut connaitre le nouvel emplacement du repère principal de l'objet aprés le déplacement. Comme les points de départ et d'arrivée du centre de la pince (les 2 extrémités du tube) sont exprimés en absolu, nous pouvons en déduire la transformation qu'il faut faire subir au point de départ pour avoir le point d'arrivée.

En effet, d'aprés Paul : $R\text{arrivée}_{(R0)} = R\text{départ}_{(R0)} \,.\, R\text{arrivée}_{(R\text{départ})}$

et donc : $R\text{arrivée}_{(R\text{départ})} = (R\text{départ}_{(R0)})^{-1} \,.\, R\text{arrivée}_{(R0)}$

il suffit alors d'appliquer cette transformation au repère principal de l'objet pour avoir sa position absolue aprés le déplacement :

$$RP\text{-après-dépl}_{(R0)} = RP\text{-avant-dépl}_{(R0)} \,.\, R\text{arrivée}_{(R\text{départ})}$$

Notons que, puisque nous avons considéré la pince comme un parallélépipède, elle sera également répértoriée en tant qu'objet dans la B.D. et nous devrons donc également mettre à jour sa position. Grâce au fait que nous considérons la pince comme n'importe quel objet et que son nom apparaît dans la primitive, il sera très aisé de considérer plusieurs pinces dans l'espace de travail et de travailler avec elles pour autant que le niveau "XYZ" le permette.

5.8. ACTIONS SUR LES OBJETS ET LES CIRCUITS.

5.8.1. DEPLACEMENT D'UN OBJET DANS UN CIRCUIT.

La primitive aura, en entrée, le nom d'un objet, le nom de la pince qui déplace l'objet (éventuellement non acompagnée de nom d'objet si elle est seule à se déplacer) et de 2 points dans l'espace (départ et arrivée). On considérera également que si un nom d'objet est présent, celui-ci aura été préalablement saisi par la pince.

Nous devrons donc construire un circuit grâce à la primitive vue précédemment et ensuite parcourir chaque tube de celui-ci grâce à la primitive de déplacement d'objet dans un tube.

La grosse difficulté réside à résoudre le problème déja evoqué : trouver les valeurs des diamètres adéquats dans chacune des 2 primitives qui seront utilisées:

- Pour la primitive de déplacement dans un tube, le diamètre du tube sera égal à la tolérance du tube de niveau "XYZ".
- Pour la primitive de construction d'un circuit, le diamètre du tube sera égal à la tolérance du tube de niveau "XYZ" plus la taille de l'objet.

Les problèmes suivants se posent :

1. Déterminer la taille de l'objet (en fait, trouver le rayon d'une sphère enveloppant totalement l'objet).

2. Fixer la tolérance du tube de niveau "XYZ" . En effet, répétons que de la tolérance que l'on aura donné à ce tube dépendra directement la performance du mouvement effectué, ceci aux dépens de la recherche d'un circuit sans collision (plus on donnera une grande tolérance, plus le mouvement sera performant, plus on aura de chance de causer une collision).

Ces deux problèmes doivent encore être résolus.

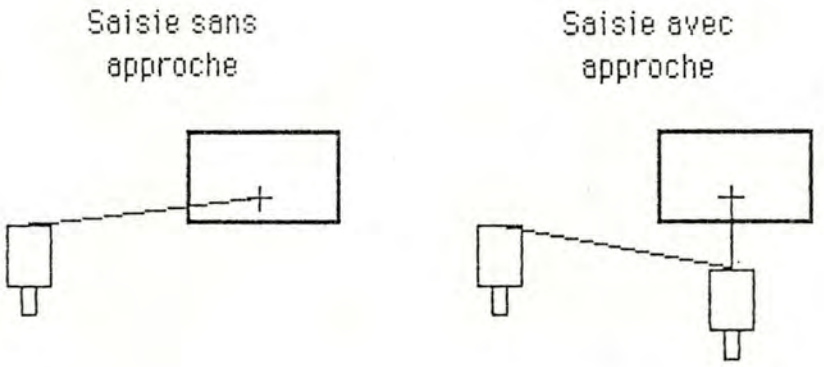
5.8.2. SAISIE D'UN OBJET.

Avec, au départ, le nom d'un objet et d'une pince, la primitive aura pour effet de saisir l'objet cité avec la pince demandée.

Cette primitive n'a pas non plus été implémentée bien qu'elle paraisse, à priori, ne pas poser trop de problèmes (Pour autant qu'il existe une primitive de déplacement d'objets). Les seuls points délicats la concernant et que nous n'avons pas encore abordés sont :

1. Déterminer un voxelor de saisie optimal parmi les voxelors de l'ensemble des parallélépipèdes de saisie possibles de l'objet.

2. Déterminer un voxelor d'approche. En effet, il faut veiller à ce que, lors du déplacement de la pince jusqu'au voxelor de saisie choisi, elle ne heurte l'objet qu'elle veut saisir en parcourant la trajectoire directe . Il faut donc déterminer un point d'approche afin d'éviter la collision entre la pince et l'objet :



Pour implémenter la primitive de saisie il faudra donc etablir la séquence d'actions suivante :

- déterminer les parallélepipèdes de saisie de l'objet : par la primitive de recherche des points de saisie d'un objet.
- choisir un voxelor de saisie "idéal" parmis l'ensemble des parallélépipèdes de saisie proposés.
- déterminer un point d'approche.
- deplacer la pince j'usqu'au point d'approche : par la primitive de déplacement d'un objet dans un circuit.
- déplacer la pince du point d'approche au point de saisie. On utilisera ici, la primitive de déplacement d'un objet dans un tube pour pouvoir descider du diamètre à employer pour le tube et donc avoir un certain pouvoir de déscision sur la manière dont doit s'effectuer l'approche.
- fermer la pince

6. CONCLUSION.

Le but initial de ce mémoire était d'implémenter une couche de programmation "objet" sur la couche "XYZ" déjà existante. Cette couche doit offrir un niveau de programmation robotique évolué au sens où il doit permettre à l'utilisateur de spécifier la tâche à accomplir en termes de buts décrivant des relations spatiales entre objets plutôt qu'en termes d'orientations que doit prendre l'outil pour réaliser ces buts. Ainsi, le programmeur pourra décrire sa tâche de manière plus naturelle en spécifiant le travail à accomplir plutôt que la façon dont il faut l'accomplir.

Pour implémenter la couche objet, il est rapidement apparu qu'il faudrait représenter les différents objets contenus dans l'espace de travail du robot. Il a donc fallu créer un modèle de représentation de solides en 3 dimensions qui soit adapté à l'utilisation que l'on voulait en faire, c'est-à-dire, assez complexe pour pouvoir faire des représentations qui soient le plus proche possible de la réalité tout en restant simple dans sa structure de base et ses concepts: Ainsi, ce modèle pourra être implémenté et utilisé plus facilement et plus rapidement.

Nous avons donc basé la structure du modèle sur une décomposition hiérarchique dont la relation entre les niveaux est une relation de complexité croissante des représentations, pour avoir des descriptions d'objets de plus en plus précises et de plus en plus proches de la réalité au fur et à mesure que l'on se rapproche du sommet de la hiérarchie (voxelor, objet primitif, objet, objet complexe). A chacun de ces niveaux sont associées des primitives afin de pouvoir manipuler les objets décrits. Ces primitives utilisent celles des niveaux inférieurs et deviennent de plus en

plus riches et puissantes en fonction du niveau auquel elles appartiennent. Enfin, pour ne pas trop nous disperser dans l'élaboration de séries de primitives de bas niveau qui auraient pu ne jamais servir aux niveaux supérieurs, nous avons concentré notre travail sur deux sujets "brûlants" en robotique : la recherche de trajectoires sans collision, et surtout la saisie automatique d'objets. Nous avons donc parcouru la hiérarchie de bas en haut en gardant une base minimale de telle sorte qu'elle déserve essentiellement les besoins des niveaux supérieurs.

Prolog a été utilisé comme langage de programmation pour réaliser les applications décrites précédemment. Au départ, l'emploi de Prolog était imposé; il fallait en effet que les applications décrites dans la couche objet soient compatibles avec celles composant la couche "XYZ", écrites également en Prolog. De plus, l'attrait d'apprendre à utiliser un nouveau langage qui prend de plus en plus d'importance dans l'informatique actuelle n'était pas négligeable. Cependant, l'innovation à ce niveau n'a pas été d'apprendre un nouveau langage mais bien d'apprendre une nouvelle façon de programmer. Il a évidemment fallu un assez long temps d'adaptation à une méthode de programmation fortement éloignée de celle utilisée par les langages Pascal-like habituels, mais une fois acquises certaines habitudes et visions particulières à Prolog, il apparaît qu'un tel langage s'adapte généralement harmonieusement à la résolution d'un problème.

Bien sûr, Prolog n'est pas du tout adapté actuellement aux traitements dont nous avions besoin (calcul matriciel, calcul trigonométrique), et les performances des programmes sont évidemment nettement insuffisantes. Cela n'a cependant pas tellement d'importance

puisque nous n'espérions pas implémenter quelque chose de réellement performant et nous pouvons espérer que Prolog devienne plus compétitif dans de tels domaines.

L'ennui majeur de ce langage est d'être très jeune et donc de faire l'objet de modifications et d'améliorations continuelles. Les programmes de cette application ont été écrits en LPA Mac Prolog sur Macintosh, et malheureusement, cet interpréteur conçu par Mac Cabe en 1985 en est à ses premières versions; il comporte encore pas mal d'erreurs et d'imprévus (tels que, par exemple des prédicats prédéfinis dans le manuel mais non implémentés) qui ont considérablement retardé l'implémentation des primitives associées aux différentes composantes de notre modèle.

La série de primitives que nous avons développée au chapitre précédent n'est certainement pas une liste exhaustive d'actions que l'on peut faire sur le modèle proposé. Le niveau "objet complexe" du modèle n'a pas du tout été abordé et peut certainement faire l'objet d'actions qui seraient dans la ligne de celles que nous avons décrites (déplacement d'un objet complexe en respectant ses contraintes...). De plus, grâce à sa structure hiérarchique, le modèle peut facilement être completé par nouvelles actions qui viendraient se greffer à ses différents niveaux en fonction des besoins (assembler deux objets, déposer un parallélépipède sur un autre...). On pourrait également continuer à l'enrichir en lui greffant de nouveaux objets primitifs (sphères, cylindres,... et pourquoi pas rombododécahédron!...) et en ajoutant des actions sur ces nouveaux objets primitifs. Ou même, il serait possible de mettre de nouveaux niveaux au sommet de la primitive pour pouvoir raffiner davantage la description des objets (par exemple un niveau supplémentaire qui donnerait les caractéristiques physiques des objets tel

que leur poids, la position de leur centre de gravité, leur niveau de fragilité,...).

La plus grosse lacune du modèle proposé est probablement de ne tenir compte des capteurs et des senseurs qui pourraient se connecter au robot. L'introduction de senseurs simples au sein du modèle (senseurs binaires) se ferait sans doute sans trop de difficultés mais la gestion de senseurs plus complexes (senseurs de force, caméra,...) est réellement très délicate à élaborer. Cette absence de possibilité d'utilisation de senseurs diminue certainement beaucoup la valeur du modèle.

De plus, le modèle proposé comporte nombre d'hypotheses simplificatrices : au point de vue de la modélisation des objets, nous n'avons considéré que les parallélépipèdes rectangles comme "objet primitif" et ils ne permettent de se combiner qu'au seul moyen de l'opérateur ensembliste d'union pour former les "objets"; au point de vue des primitives travaillant sur ces parallélépipèdes et de ces compositions de parallélépipèdes, bien des simplifications ont également été faites; tant du point de vue des parcours de trajectoires sans collisions que du point de vue de la saisie automatique. Nous ne les rappellerons pas ici etant donné qu'elles sont souvent fort techniques; ce rappel demanderait de restituer tout le contexte pour rendre les choses compréhensibles.

Ces hypothèses simplificatrices ont sans aucun doutes très fort diminué les performances du modèle et ses possibilités d'utilisation. Pourtant, on a pu se rendre compte de la grande portée et de la souplesse des concepts que l'on a utilisés. Par exemple, avec les seuls concepts de parallellepipèdes et de primitive de recouvrement, on a pu aussi bien gérer

bien la recherche de trajectoire sans collision que la détermination des points de saisie (en générant de nouveaux parallélépipèdes de saisie et enveloppe et en employant la primitive de recouvrement sur ces nouveaux concepts).

Le modèle proposé offre un intérêt supplémentaire : les concepts qu'il met en oeuvre sont assez facilement exprimés par des éléments mathématiques fort souples, et d'utilisation relativement aisée (calcul matriciel, trigonométrie, géométrie dans l'espace,...). Malheureusement, même si ces concepts sont assez simples en eux-même, ils sont parfois difficiles à utiliser pour un opérateur humain à cause de la difficulté de visualisation d'objets en 3 dimensions à partir de leur modélisation mathématique. En effet, s'il est encore assez facile de déterminer la matrice et les dimensions d'un parallélépipède que l'on veut représenter, la vérification des résultats d'un programme fournissant la liste des parallélépipèdes de saisie de ce parallélépipède est une tâche ardue. Il aurait été beaucoup plus aisé de pouvoir tester les programmes en vérifiant les résultats à l'aide d'un logiciel de graphisme tri-dimensionnel. Un tel logiciel pourrait d'ailleurs servir de base pour un logiciel de construction interactive d'objets à représenter (et donc d'intermédiaire à un langage de description d'objets), ou pour un logiciel de simulation du robot lors de l'exécution d'un programme.

Ces problèmes de visualisation sont liés à l'existence même de modèles de représentation géométrique de l'espace à 3 dimensions. Comme le lecteur a certainement pu s'en rendre compte, il est généralement très difficile de travailler dans une représentation d'espace à trois dimensions. Des concepts qui paraissent fort simples dans la réalité se révèlent souvent très complexes quand il faut systématiser et décrire les processus qui les

manipulent. Par exemple, il n'y a rien qui paraisse plus simple que de déterminer les orientations que pourrait prendre la pince pour saisir un objet quand on se trouve à côté de l'objet et que l'on simule la pince avec sa main. Pourtant, le même probléme se révèle rapidement un casse-tête quand on se préoccupe de trouver un processus mathématique qui permette de générer automatiquement ces orientations.

REFERENCES :

[ALB84] Albus J.S., "Robotics", in "Robotics and A.I", NATO ASI series, pp 65-94, 1984

[AMB83] Ambler A.P., "Languages for programming robots", D.A.I research paper 208, Univ. Edimburgh, 1983.

[BES85] Besl P.J. & Jain R.C., "three dimensional object recognition.", ACM Computing Surveys, vol 17, pp 83-91, 1985.

[BOY79] Boyse J.W., "interference detection among solids and surfaces.", Communications of the ACM, vol 22, pp 3-9, 1979.

[BRA84] Brady M., "Representing shape", in "Robotics and A.I", NATO ASI series, pp 279 - 300, 1984

[BRA85] Brady M., "Artificial intelligence and robotics", Artificial Intelligence, vol 26, pp 79-121, 1985.

[COI81] Coiffet Ph., "Les Robots - Tome 1 : Modélisation et commande", Hermès, 1981.

[FAU84] Faugeras O.D., Hebert M., Pauchon E. & Ponce J., "Object representation, identification and positionning from range data", in "Robotics and A.I", NATO ASI series, pp 255-277, 1984

[GRO76] Grossman D.D., "Procedural representation of three dimentional objects.", IBM Journal Res. Dev., vol 20, pp 582-589, 1976.

[HAS79] Hasegawa T. & Inoue H., "Modelling and monitoring a manipulator environment", Proceedings on the Sixth International Joint Conference on A.I., Tokyo, Volume 1, pp 369 - 371, August 1979.

[LAT84] Latombe J-C., "Automatic synthesis of robot programs from CAD specifications", in "Robotics and A.I", NATO ASI series, pp 3-47, 1984.

[LAM83] Lambird B.A., "High-level languages for spatial knowledge.", Computer science technical report series, University of Maryland TR-1275, pp 43-45, 1983.

[LIE77] Lieberman L.I & Wesley M.A., "Autopass : an automatic programming system for computer controlled mechanical assembly.", IBM Journal Dev., vol 21, pp 3212-333, 1977.

[LOZ79] Lozano-perez T. & Wesley M.A., "An algorithm for planning collision-free paths among polyhedral obstacles.", Communications of the ACM, vol 22, pp 560-570, 1979.

[LOZ81] Lozano-Pérez T, "Automatic planning of manipulator transfer movements", IEEE Trans. Syst., Man, Cybern., vol SMC11, pp 681-698, 1981.

[LOZ85] Lozano-Pérez T, "Compliance in robot manipulation", Artificial Intelligence, vol 25, pp 5-12, 1985.

[NAG84] Nagel R., "State of the art and predictions for A.I and robotics", in "Robotics and A.I", NATO ASI series, pp 3-47, 1984.

[PAU81] Paul R.P., "Robot manipulators - Mathematics, Programming & Control", MIT, 1981.

[PER83] Perricone B.T., "spatial representation and reasoning", Computer science technical report series, University of Maryland, TR-1275, pp 2-17, 1983.

[REQ80] Requicha A.A.G, "Representation for rigids solids : theory, methods and systems.", ACM Computer Survey, vol 12, pp 437-464, 1980.

[REQ83] Requicha A.A.G & Voelker M.B., "Solid modeling : current status and research directions.", IEEE computer graphics and applications, october 1983, pp25-37.

[TIL84] Tilove R.B., "A nul-object detection algorithm for Constructive Solid Geometry.", Communications of the ACM, vol 27, pp 684-694, 1984.

[VAN85] Vanhemelryck C., "Construction d'un environenment d'aide a la programmation d'un robot.", Mémoire de licence FUNDP, 1985.

[WES80] Wesley M.A. & al, "A Geometric Modeling System for automated mechanical assembly.", IBM Journal Res. Dev., vol 21, pp 321-333, 1977.

[YAU83] Yau M. & Srihari S.N., "A hierarchical data structure for multidimensional digital images.", Communications of the ACM, vol 26, pp 504-515, 1983.

ANNEXES.

CONVENTION DE REPRESENTATION MACHINE DES OBJETS.

Voxelor sous forme matricielle:

Une matrice contenant 4 vecteurs representant chacun une colonne de la matrice.

mat(v(A11, A21, A31, A41), v(A12, A22, A32, A42),
v(A13, A23, A33, A43), v(A14, A24, A34, A44))

Voxelor sous forme rotationelle :

6 variables representant les translations (X, Y, Z) et les rotations (T, R, L) suivant les 3 axes de R0.

vox(X, Y, Z, T, R, L).

Parallelepipedes :

1 matrice 4x4 donnant le repere du parallelepipede et 3 variable donnant ses dimensions suivant les 3 axes de son repere.

paral(mat(V1, V2, V3, V4), args(Lx, Ly, Lz))

Objets :

Liste de parallelepipedes. La premiere matrice du premier parallelepipede sera le repere principal de l'objet, et donc, les matrices des autres parallelepipede seront exprimees en fonction de cette premiere matrice.

[paral(Mat1 , Args1) , ... ,paral(MatN , ArgsN)]

```
transfo_matrice_en_rot( mat(A, 0, N, v(X,Y,Z,1)), vox(X,Y,Z,T,R,L)) :-
                langage( N, T),
                rouinst( N, T, R),
                facet( A, 0, T, L),!

langage( v(Nx, Ny, Nz,0), 0) :-
                arctg2( Nx, Nz, indef)

langage( v(Nx, Ny, Nz,0), T) :-
                arctg2( Nx, Nz, T)

rouinst( v(Nx, Ny, Nz,0), T, R) :-
                sin(T, St),    cos( T, Ct),
                k   is   St * Nx  +  Ct * Nz,
                J   is    -Ny,
                arctg2( J, k, R)

facet( v(Ax, Ay, Az,0), v(0x, 0y,0z,0), T, L) :-
                sin(T, St),    cos( T, Ct),
                J   is   St * 0z  -  Ct * 0x,
                k   is   Ct * Ax  -  St * Az,
                arctg2( J, k, L)
```

```
transfo_rot_en_matrice(vox(X,Y,Z,T, R, L),
                       mat(v(Ax, Ay, Az,0), v(Ox, Oy, Oz,0), v(Nx, Ny,Nz,0),v(X,Y,Z,1))) -
                  sin( T, St), cos( T, Ct),
                  sin( R, Sr), cos( R, Cr),
                  sin( L, Si), cos( L, Ci),
                  Ox is ( St * Sr * Ci ) - ( Ct * Si ),
                  Oy is ( Cr * Ci ),
                  Oz is ( Ct * Sr * Ci ) + ( St * Si ),
                  Nx is ( Si * Cr ),
                  Ny is ( - Sr ),
                  Nz is ( Ct * Cr),
                  proaut( v(Ox,Oy,Oz), v(Nx,Ny,Nz), v(Ax,Ay,Az))

proaut(v(A1,A2,A3),v(O1,O2,O3), v(N1,N2,N3)) :-
        N1 is A2*O3 - A3*O2.
        N2 is A1*O3 - A3*O1.
        N3 is A1*O2 - A2*O1.
```

```
arctg2(0,0,Angle)
arctg2(Y,0,90)      - >(Y,0)
arctg2(Y,0,270)     - <(Y,0)
arctg2(0,X,0)       - >(X,0)
arctg2(0,X,180)     - <(X,0)
arctg2(Y,X,Angle)   - X > 0, Y > 0,  Z is Y/X, tg(Z,Angle)
arctg2(Y,X,Angle)   - X > 0, Y < 0,  Z is Y/X, tg(Z,A1), Angle is A1 + 180
arctg2(Y,X,Angle)   - X < 0, Y > 0,  Z is Y/X, tg(Z,Angle)
arctg2(Y,X,Angle)   - X < 0, Y < 0,  Z is Y/X, tg(Z,A1), Angle is A1 + 180

tg(Z,Angle) - write('Pas impiemente')

sin(0,0)
sin(60,0.7)
sin(90,1)
sin(180,0)
sin(270,-1)
sin(360,0)

cos(0,1)
cos(60,0.5)
cos(90,0)
cos(180,-1)
cos(270,0)
cos(360,1)
```

```
/*
   distinct(P1,P2) reussi si l'intersection de volume entre les 2
   parallelepipedes P1 et P2 est vide. Si les 2 paral. sont  imbriques
   - meme partiellement - , le predicat echoue

   methode employee : application de : Si un des 2 paral a au moins
    deux faces paralelles telles que tous les coins de l'autre paral se
    trouvent du meme cote des 2 faces , alors il n'y a pas overlap entre
    ces 2 paral
*/



distinct(P1,P2) :-
     plans(P1 , Plans_P1),
     points(P2 , Points_P2) ,
     uniat_distinct( Plans_P1 , Points_P2)

distinct(P1,P2) :-
     plans(P2 , Plans_P2),
     points(P1 , Points_P1) ,
     uniat_distinct( Plans_P2 , Points_P1)

uniat_distinct([[P1,P2]|_],_] , Points) :-
      meme_cote(P1,P2, Points).

uniat_distinct([_,[P3,P4]|_] , Points) :-
      meme_cote(P3,P4, Points)

uniat_distinct([_,_,[P5,P6]] , Points) :-
      meme_cote(P5,P6, Points)

meme_cote(P1,P2, Liste_points) :-
      nuit_list(P1 , Liste_points , R1),
      nuit_list(P2 , Liste_points , R2),
      meme_signe( R1 , R2 )

nuit_list( Pi , [] , [] ).
nuit_list( Pi , [Pt|L_pt] , [Res|Lres] ) :-
    multiply( Pi , Pt , Res ),
    nuit_list( Pi , L_pt , Lres )

multiply(pl(A,B,C,D) , pt(P1,P2,P3,P4) , Res ) :-
```

```
    Res is A*P1 + B*P2 + C*P3 + D*P4.

meme_signe( [] , [] ).
meme_signe( [H1 | T1] , [H2 | T2] ) :-
      SIGN( H1 , SH1 ) , SIGN( H2 , SH2),
      SH1 = SH2 ,
      meme_signe( T1 , T2 ).

/* ------------------------------------------------------------------
  calcul de la coordonnee absolue des 8 coins d'un parai.
*/

points( P , Liste_points ) :-
      FSALL( Liste_points , Pt_calcule, [calcul_pt , P , Pt_calcule])


calcul_pt([Repere, args(Lx,Ly,Lz)] , Pt_calcule) :-
      calc( Repere, v(0,0,0) ,Pt_calcule).

calcul_pt([Repere, args(Lx,Ly,Lz)] , Pt_calcule) :-
      calc( Repere, v(0,0,Lz) ,Pt_calcule).

calcul_pt([Repere, args(Lx,Ly,Lz)] , Pt_calcule) :-
      calc( Repere, v(0,Ly,0) ,Pt_calcule).

calcul_pt([Repere, args(Lx,Ly,Lz)] , Pt_calcule) :-
      calc( Repere, v(0,Ly,Lz) ,Pt_calcule).

calcul_pt([Repere, args(Lx,Ly,Lz)] , Pt_calcule) :-
      calc( Repere, v(Lx,0,0) ,Pt_calcule).

calcul_pt([Repere, args(Lx,Ly,Lz)] , Pt_calcule) :-
      calc( Repere, v(Lx,0,Lz) ,Pt_calcule).

calcul_pt([Repere, args(Lx,Ly,Lz)] , Pt_calcule) :-
      calc( Repere, v(Lx,Ly,0) ,Pt_calcule).

calcul_pt([Repere, args(Lx,Ly,Lz)] , Pt_calcule) :-
      calc( Repere, v(Lx,Ly,Lz) ,Pt_calcule).


calc( mat(v(X1,X2,X3,0), v(Y1,Y2,Y3,0), v(Z1,Z2,Z3,0), v(Dx,Dy,Dz,1)) ,
          v(Lx,Ly,Lz) , pt(P1,P2,P3,P4) ) :-
```

```
        P1  is  X1*Lx + Y1*Ly + Z1*Lz + Dx ,
        P2  is  X2*Lx + Y2*Ly + Z2*Lz + Dy ,
        P3  is  X3*Lx + Y3*Ly + Z3*Lz + Dz ,
        P4  is  1

/*_____________________________________________________________________

  calcul des coefficients des equations des 6 plans determinant le volume d'un parai.
*/

plans( P , Liste_plans) :-
        ISALL(Liste_plans , Plans_paralelles , [calcul_pl , P , Plans_paralelles] )

calcul_pl( [mat(V1,_,_,V4) , args(Lx,_,_)] , Plans_paralelles ) :-
        calc2(V1,V4,Lx , Plans_paralelles ).

calcul_pl( [mat(_,V2,_,V4) , args(_,Ly,_)] , Plans_paralelles ) :-
        calc2(V2,V4,Ly , Plans_paralelles ).

calcul_pl( [mat(_,_,V3,V4) , args(_,_,Lz)] , Plans_paralelles ) :-
        calc2(V3,V4,Lz , Plans_paralelles ).

calc2(v(A,B,C,0), v(Dx,Dy,Dz,1) , L , [pl(A,B,C,D1),pl(A,B,C,D2)]) :-
        D1  is  -(A*Dx) - (B*Dy) - (C*Dz) ,
        D2  is  D1 - L
/*_____________________________________________________________________
```

```
/*
cherche la liste des matrices de transformation TRpa decrivants les differentes orientations
que peut prendre la pince pour saisir le parai.
ces orientations seront exprimees en fonction du repere du paraleleipipede
*/


rech_matrices_de_transformation(args( Lx, Ly, Lz ), args( _ , Lo, _ ), Liste_mat) :-
        mat_transfo_cote(mat(v(1,0,0),v(0,1,0),v(0,0,1)), Lx, Lo, Liste1),
        mat_transfo_cote(mat(v(0,1,0),v(1,0,0),v(0,0,1)), Ly, Lo, Liste2),
        mat_transfo_cote(mat(v(0,0,1),v(1,0,0),v(0,1,0)), Lz, Lo, Liste3),
        append(Liste1, Liste2, Liste12),
        append(Liste12, Liste3, Liste_mat).


mat_transfo_cote(mat(V1,V2,V3), L, Lo, [] ) :-
        =:(L, Lo).

mat_transfo_cote(mat(V1,V2,V3), L, Lo, Liste_mat_face) :-
        =:(L, Lo),
        !SALL( Liste_mat_face , Mat_transfo ,
                [calcul_mat , V1,V2,V3,Mat_transfo])


calcul_mat( V1,V2,_ , mat(V2,V1,N)) :-
        produit( V2,V1 , N )

calcul_mat( V1,V2,_ , mat(MoinsV2,V1,N)) :-
        inv_signe(V2 , MoinsV2 ),
        produit( MoinsV2,V1 , N )

calcul_mat( V1,_,V3 , mat(V3,V1,N)) :-
        produit( V3,V1 , N )

calcul_mat( V1,_,V3 , mat(MoinsV3,V1,N)) :-
        inv_signe(V3 , MoinsV3 ),
        produit( MoinsV3,V1 , N )

calcul_mat( V1,V2,_ , mat(V2,MoinsV1,N)) :-
        inv_signe(V1 , MoinsV1 ),
        produit( V2,MoinsV1 , N )
```

```
calcul_mat( V1,V2_ , mat(MoinsV2,MoinsV1,N)) -
        inv_signe(V1 , MoinsV1 ),
        inv_signe(V2 , MoinsV2 ),
        produit( MoinsV2,MoinsV1 , N )

calcul_mat( V1_,V3 , mat(V3,MoinsV1,N)) -
        inv_signe(V1 , MoinsV1 ),
        produit( V3,MoinsV1, N ).

calcul_mat( V1_,V3 , mat(MoinsV3,MoinsV1,N)) -
        inv_signe(V1 , MoinsV1 ),
        inv_signe(V3 , MoinsV3 ),
        produit( MoinsV3,MoinsV1 , N )

inv_signe(v(A1,A2,A3), v(B1,B2,B3)) -
        B1 is -A1,
        B2 is -A2,
        B3 is -A3

produit(v(A1,A2,A3),v(01,02,03), v(N1,N2,N3)) -
        N1 is A2*03 - A3*02,
        N2 is A1*03 - A3*01,
        N3 is A1*02 - A2*01.
```

```
/*
A partir de la matrice de transfo, des dimension du parai à saisir, et des  dimensions de la
pince qui doit le saisir, rech_psaisie_rel(Mat_transfo, Args_parai, Args_pince, Pos , Args)
rend la coordonee et les dimensions du parai de saisie. La coordonee sera en fonction du repere
du parai à saisir
*/


rech_psaisie_rel(Mat_transfo, args(Lx, Ly, Lz), Args_pince,
                                pos(X,Y,Z), args(Lsx,Lsy,Lsz)) -
        translation_et_longeur(v(1,0,0), Lx, Args_pince,
                                Mat_transfo,X,Lsx,Lsy,Lsz),
        translation_et_longeur(v(0,1,0), Ly, Args_pince,
                                Mat_transfo,Y,Lsx,Lsy,Lsz),
        translation_et_longeur(v(0,0,1), Lz, Args_pince,
                                Mat_transfo,Z,Lsx,Lsy,Lsz).

translation_et_longeur(V, Lv, Args_pince, Mat_transfo, Dv, Lsx, Lsy, Lsz) -
        vecteur_paralelle(V, Mat_transfo, V2, Sens),
        trans_long(V2, Sens, Lv, Args_pince, Dv, Lsx, Lsy, Lsz)

vecteur_paralelle(V, mat(V,0,N), a, 1)
vecteur_paralelle(V, mat(A,V,N), o, 1).
vecteur_paralelle(V, mat(A,0,V), n, 1)
vecteur_paralelle(v(X1,Y1,Z1), mat(A,0,N), V2, Sens) -
        X2 is -X1 ,  Y2 is -Y1 ,  Z2 is -Z1,
        vecteur_paralelle(v(X2,Y2,Z2), mat(A,0,N), V2, Sc),
        Sens is -Sc

trans_long( a, 1, Lv, args(LpinceX, _ , _ ), Dv, Lsx, Lsy, Lsz) -
        >( Lv, LpinceX ),
        Dv  is  LpinceX / 3,
        Lsx is ( 2 * LpinceX) / 3

trans_long( a, -1, Lv,args(LpinceX, _ , _ ), Dv, Lsx, Lsy, Lsz) -
        >( Lv, LpinceX ),
        Dv  is  Lv - (LpinceX / 3),
        Lsx is ( 2 * LpinceX) / 3
```

```
trans_iong( a, _ , Lv, Args(LpinceX, _ , _ ), Dv, Lsx, Lsy, Lsz) :-
            =:( Lv, LpinceX ),
            Dv  is  Lv / 2.
            Lsx is  Lv / 2

trans_iong(o, _ , Lv, _ , Dv, Lsx, Lsy, Lsz) :-
            Dv  is  Lv / 2.
            Lsy is  1

trans_iong(n, 1, Lv, args( _ , _ , LpinceZ), Dv, Lsx, Lsy, Lsz) :-
            Dv  is  LpinceZ / 3,
            Lsz is  Lv - (( 2 * LpinceZ ) / 3 ).

trans_iong(n, -1, Lv, args( _ , _ , LpinceZ), Dv, Lsx, Lsy, Lsz) :-
            Dv  is  Lv - (LpinceZ / 3),
            Lsz is  Lv - (( 2 * LpinceZ ) / 3 )
```

```
/*
A partir d'un parallelepipede et de la pince destinee a le saisir, saisie_parai(Parai,Pince,L)
rend une liste de parallelepipede de saisie possible pour la pince
*/

saisie_parai( parai(Repere,Args) , parai(Repere_pince,Args_pince), Liste_psaisie) -
        parai( Nom_pince,[ _ , Args_pince]),
        rech_matrices_de_transformation( Args, Args_pince, Liste_mat_transfo),
        rech_liste_psaisie_rel( Liste_mat_transfo,Args,Args_pince, Liste_psaisie_rel),
        transfo_liste_psaisie_rel_en_abs(Repere , Liste_psaisie_rel, Liste_psaisie)


rech_liste_psaisie_rel( [] , _ , _ , [] ).
rech_liste_psaisie_rel( [Mat_transfo | Liste_mat_tr] , Args, Args_pince,
                        [ [Mat_psaisie_rel , Arg_psaisie] | Liste_psaisie] ) -
        rech_psaisie_rel(Mat_transfo, Args, Args_pince, pos(X,Y,Z), Arg_psaisie),
        matrice_psaisie_rel(Mat_transfo,pos(X,Y,Z), Mat_psaisie_rel),
        rech_liste_psaisie_rel( Liste_mat_tr , Args, Args_pince, Liste_psaisie)

matrice_psaisie_rel(mat(v(A1,A2,A3),v(O1,O2,O3),v(N1,N2,N3)),pos(X,Y,Z),
                    mat(v(A1,A2,A3,0),v(O1,O2,O3,0),v(N1,N2,N3,0),v(X,Y,Z,1)))


transfo_liste_psaisie_rel_en_abs( _ , [] , [] )
transfo_liste_psaisie_rel_en_abs(Mat_parai, [ [Mat_rel, Arg_psaisie] | Liste_mat_rel],
                                            [ [Mat_abs, Arg_psaisie] | Liste_mat_abs]) -
        transfo_rel_en_abs(Mat_parai,Mat_rel,Mat_abs),
        transfo_liste_psaisie_rel_en_abs(Mat_parai, Liste_mat_rel,
Liste_mat_abs)
```

```
/*
objet_distinct(O1,O2) reussi si les 2 objets O1 et O2 sont distincts s'ils sont juxtaposes,
meme partiellement, le predicat echoue.
*/

objets_distincts( [] , O2 ).
objets_distincts( [Comp1 | L_comp] , O2 ) :-
        tous_distincts( Comp1 , O2 ),
        objets_distincts( L_comp , O2 ).

tous_distincts( _ , [] ).
tous_distincts( P1 , [P2 | Liste_p] ) :-
        distincts( P1 , P2 ),
        tous_distincts( P1 , Liste_p ).
```

```
/*
saisie_objet(Objet, Pince, L) rend dans L la liste des parai de saisie possible d'un objet
*/

saisie_objet( Liste_composants , Pince , Liste_parai_de_saisie ) :-
        trt_saisie( Liste_composants , Liste_composants , Pince ,
                        Liste_parai_de_saisie )

trt_saisie( [] , _ , _ , [] ).
trt_saisie( [Parai_a_saisir | Liste_parai] , Liste_comp , Pince ,
            [Lp_de_saisie_possible | Liste_Lp_de_saisie] ) :-
        effacer( Parai_a_saisir , Liste_comp , L_comp ),
        saisie_parai( Parai_a_saisir , Pince , Lp_de_saisie ),
        parai_de_saisie_possible( Lp_de_saisie , L_comp , Pince,
                                        Lp_de_saisie_possible ),
        trt_saisie( Liste_parai , Liste_comp , Pince , Liste_Lp_de_saisie ).

effacer( X , [X|T] , T ).
effacer( X , [H|T] , [H|T2] ) :- effacer( X , T , T2 ).

parai_de_saisie_possible( [] , _ , _ , [] ).
parai_de_saisie_possible( [P_de_saisie|Lp] , L_comp , Pince,
                                        [P_de_saisie|Lp_possible] ) :-
        transfo( P_de_saisie , Pince , P_enveloppe ),
        tous_distincts( P_enveloppe , L_comp ),
        parai_de_saisie_possible( Lp , L_comp , Pince , Lp_possible ).
parai_de_saisie_possible( [P_de_saisie|Lp] , L_comp , Pince , Lp_possible ) :-
        parai_de_saisie_possible( Lp , L_comp , Pince , Lp_possible ).

tous_distincts( _ , [] ).
tous_distincts( P_enveloppe , [P_composant|L_comp] ) :-
        distinct( P_enveloppe , P_composant ).

transfo( [ Mat_saisie , args(LsX,LsY,LsZ) ] , Pince ,
                [ Mat_enveloppe , args(LeX,LeY,LeZ) ] ) :-
        parai(Pince , [ _ , args( LpinceX , LpinceY , LpinceZ )]) ,
        Dx is -LpinceX  ,  Dy is -(LpinceY / 2)  ,  Dz is -(LpinceZ / 2) ,
        transfo_rei_en_abs( Mat_saisie ,
                        mat(v(1,0,0,0),v(0,1,0,0),v(0,0,1,0),v(Dx,Dy,Dz,1)),
                        Mat_enveloppe ) ,
        LeX is LsX + LpinceX ,
        LeY is LpinceY ,
```

LeZ is LsZ.

```
/*
 Construction d'un barai comprenant un tube de diametre D et dont les 2
 extremites sont donnes par 2 voxelors.
*/

construction_tube(mat( _ , _ , _ , Tr1) , mat( _ , _ , _ , Tr2),
                  D , [mat(Vx,Vy,Vz,Vd) , Args]) :-
        constr_vx( Tr1 , Rr2 , Vx ),
        constr_vy( Vx , Vy ),
        constr_vz( Vx , Vy , Vz ),
        constr_vd( Tr1 , Vy , Vz , D , Vd ),
        constr_args( Tr1 , Tr2 , D , Args )


constr_vx(v(Dx1,Dy1,Dz1,1), v(Dx2,Dy2,Dz2,1), Vx) :-
        A is Dx2 - Dx1 , B is Dy2 - Dy1 , C is Dz2 - Dz1,
        normalisation(v(A,B,C,0) , Vx).

constr_vy(v(Ax,Bx,Cx,0), Vy) :-
        not 'E0'(Cx,0),
        C is -(Bx/Cx),
        normalisation(v(0,1,C,0) , Vy )

constr_vy(v(Ax,Bx,Cx,0), Vy) :-
        not 'E0'(Bx,0),
        C is -(Cx/Bx),
        normalisation(v(0,B,1,0) , Vy ).

constr_vy(v(Ax,Bx,Cx,0), Vy) :-
        not 'E0'(Ax,0),
        C is -(Cx/Ax),
        normalisation(v(0,1,C,0) , Vy )


constr_vz(v(Ax,Bx,Cx,0),v(Ay,By,Cy,0),Vz) :-
        A is Bx*Cy - Cx*BY,
        B is Ax*Cy - Cx*Ay,
        C is Ax*By - Bx*Ay,
        normalisation(v(A,B,C,0) , Vz).

constr_vd(v(Dx1,Dy1,Dz1,1) , v(Ay,By,Cy,0),
          v(Az,Bz,Cz,0) , D , v(Dx,Dy,Dz,1)) :-
```

```
        Dx is Dx1 - ((Ay+Az)*D/2),
        Dy is Dy1 - ((By+Bz)*D/2),
        Dz is Dy1 - ((Cy+Cz)*D/2)

constr_args(Tr1 , Tr2 , D , args(Lx,D,D)) :-
        distance(Tr1 , Tr2 , Lx)

distance(v(X1,Y1,Z1,_), v(X2,Y2,Z2,_), Dist) -
        X is X2 - X1 , Y is Y2 - Y1, Z is Z2 - Z1,
        'SQRT'(DX, X),
        'SQRT'(DY, Y),
        'SQRT'(DZ, Z),
        D2 is DX + DY + DZ,
        'SQRT'(D2, Dist).

normalisation(v(A,B,C,0), v(Ax,Bx,Cx,0)) -
        'ABS'(A,AbsA),'ABS'(B,AbsB),'ABS'(C,AbsC),
        N is AbsA + AbsB + AbsC,
        Ax is A/N, Bx is B/N, Cx is C/N.
```

```
parcours( Depart , Arrivee , Diametre , Pts_passage , [Tube] ) :-
        construction_tube( Depart , Arrivee , Diametre , Tube ) ,
        tube_sans_collision(Tube).

parcours( Depart , Arrivee , Diametre , Pts_passage , Parcours ) :-
        pt_via( V ),
        absent( V , Pts_passage ),
        parcours( Depart , V , Diametre , [V | Pts_passage] , P1 ) ,
        parcours( V , Arrivee , Diametre , [V | Pts_passage] , P2 ) ,
        append( P1 , P2 , Parcours )

parcours( _ , _ , _ , [] ).

append( [] , X , X ).
append( [H|T1] , L , [H|T2] ) :- append( T1 , L , T2 ).

absent( V , [] ).
absent( V , [H | T] ) :- not 'EQ'( H , V ) , absent( V , T ).

pt_via(v1).
pt_via(v2).
pt_via(v3).
```